ADVANCES IN SHORTEST PATH BASED COLUMN GENERATION FOR INTEGER PROGRAMMING

A Thesis
Presented to
The Academic Faculty

by

Faramroze Godrej Engineer

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Industrial and Systems Engineering

Georgia Institute of Technology
August 2009

ADVANCES IN SHORTEST PATH BASED COLUMN GENERATION FOR INTEGER PROGRAMMING

Approved by:

Prof. George L. Nemhauser,
Co-Advisor
School of Industrial and Systems Engineering
Georgia Institute of Technology

Prof. Martin W. P. Savelsbergh,
Co-Advisor
School of Industrial and Systems Engineering
Georgia Institute of Technology

Prof. Pınar Keskinocak
School of Industrial and Systems Engineering
Georgia Institute of Technology

Prof. Joel S. Sokol
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Kevin C. Furman
Supply Chain Optimization Team Leader
ExxonMobil Upstream Research Company

Date Approved: 18$^{\text{th}}$ June, 2009

ACKNOWLEDGEMENTS

I want to thank my advisors George Nemhauser and Martin Savelsbergh for their guidance and support. I am honoured and privileged to have worked with researchers of such high calibre. The lessons I have learnt through them will undoubtedly hold me in good stead throughout my professional and personal endeavours. I would also like to thank Ellis Johnson for his guidance during the early stages of my PhD, and Pınar Keskinocak for the opportunity to work on one of the most rewarding projects I have had the privilege to work on.

Among industry partners, I would like to thank Bob Spaulding and Bruce Sawhill of DayJet Corporation, Jin-Hwa Song and Kevin Furman of ExxonMobil Research and Engineering, and Larry Pickering of the Centers of Disease Control and Prevention. I am also grateful for the wonderful friendships that have helped me cope with the stresses of PhD life. In particular, I would like to thank Gizem Keysan, Daniel Espinoza, Ricardo Fukasawa, Renan Garcia, James Luedtke, Juan Morales, Amandeed Parmar, and Juan-Pablo Vielma.

I thank my brothers for keeping me grounded and reminding me of the important things in life. Last, but not least, I am eternally grateful to my parents for their unconditional and unwavering support, and the sacrifices they've endured to afford their sons the opportunities they could not.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Column generation is a pricing scheme for solving large scale *linear programming* (LP) problems. This process involves introducing variables corresponding to violated dual constraints dynamically during the solution process of the LP. When embedded in a branch-and-cut algorithm, the resulting branch-price-and-cut algorithm can be used to solve many difficult and large scale *integer programming* (IP) problems.

Branch-price-and-cut algorithms are among the most successful exact optimization approaches for solving many routing and scheduling problems. This is due, in part, to the availability of extremely efficient and effective dynamic programming algorithms for solving the pricing problem, as well as the availability of efficient and effective branching schemes and cutting planes that drive integrality. In terms of branch-price-and-cut, two obstacles we face today are (1) being able to solve harder and larger pricing problems, and (2) solving mixed-integer column generation formulations that suffer from relatively weak LP bounds compared to the more traditional 0-1 set partitioning type. As part of the work presented in this thesis, we encounter column generation formulations motivated by real life problems that require overcoming both types of challenges.

The first part of this thesis is dedicated to solving the *resource constrained shortest path problem* (RCSPP) arising in column generation pricing problems for formulations involving extremely large networks and a huge number of local resource constraints. We present a relaxation-based dynamic programming algorithm that alternates between a forward and a backward search. Each search employs bounds derived in

the previous search to prune the search, and between consecutive searches, the relaxation is tightened using a set of critical resources and a set of critical arcs over which these resources are consumed. The algorithm is tested on practical instances of the dial-a-flight problem and is shown to significantly outperform standard dynamic programming techniques.

The second part of this thesis also focuses on the pricing problem. In many situations, incorporating relevant practical constraints results in pricing problems that do not give rise to pure RCSPPs, but more complex variants, such as the *fixed charge shortest path problem* (FCSPP) in which the amount of resource consumed is itself a continuous bounded variable. By exploiting the structure of optimal solutions to FCSPP, we design and implement a solution approach that relies on solving multiple RCSPPs, and therefore can again make use of extremely efficient and effective dynamic programming algorithms. Computational results on three different classes of problems are reported including the split delivery vehicle routing problem, parallel machine scheduling problems with controllable processing times, and multi-period inventory routing problems.

In the third and final part of this thesis, we present a new branch-price-and-cut algorithm for the *inventory routing problem* (IRP), a difficult mixed 0-1 problem that requires considerable effort in preprocessing, branching, and cut generation outside the realms of branch-price-and-cut for traditional 0-1 problems. Apart from considering a more general problem than previously considered in the literature, we extend the algorithm developed for FCSPP to solve the pricing problem efficiently. In addition to preprocessing, we use the boundary constraints to restrict the set of columns that are generated. Furthermore, we extend a class of cuts known for the vehicle routing problem, and develop a new class of cuts specifically for IRP to tighten the formulation even further. Both the branching schemes and cuts preserve the structure

of the pricing problem making them efficiently implementable within a branch-price-and-cut algorithm. Computational results are reported for several practical instances related to managing the supply chain of a large petrochemical company. The results compare favorably against an alternative branch-and-cut algorithm.

CHAPTER I

INTRODUCTION

Column generation is the process of introducing variables corresponding to violated constraints in the dual (i.e., variables with negative reduced cost in case of minimization) dynamically during the solution process of a *linear program* (LP)

$$\begin{aligned} \min\ & cx \\ \text{s.t. } & Ax \leq\ a \\ & x \in \Re^n_+. \end{aligned}$$

This is especially useful in the context of solving LPs where the number of variables may be prohibitively large. Of course, this is only ever useful if the pricing problem (i.e., the process of finding columns corresponding to violated dual constraints) can be solved efficiently. It is hardly a surprise then that some of the seminal work in column generation relied heavily on using special purpose *dynamic programming* (DP) algorithms to solve the pricing problem whether it be for solving shortest path problems as in the work of Ford and Fulkerson [60] on the multicommodity flow problem, or solving knapsack problems as in the work of Gilmore and Gomory [67, 68] on the cutting stock problem.

Originally intended as a means for solving LPs with a large number of variables, Dantzig and Wolfe [37] recognized the potential to decompose and solve by column generation, LPs with a large number of rows but with block-diagonal structure that could be exploited. Although it would be several more years before the connection to Lagrangean relaxation, integer programming and, the integrality property would be fully revealed (see Geoffrion [66]), it did not escape them that the proposed decomposition scheme and column generation could apply more generally for decomposable

convex sets that need not necessarily be derived from LPs.

> *"Viewed as an instance of a 'generalized programming problem' whose columns are drawn freely from given convex sets, such a problem can be studied by an appropriate generalization of the duality theorem for linear programming, which permits a sharp distinction to be made between those constraints that pertain only to a part of the problem and those that connect its parts."*
>
> – George Dantzig and Philip Wolfe [37]

The middle ages of column generation (70s - 80s) saw the transition from its use as primarily an LP method to its use in integer programming (i.e., to solve LPs where some or all of the variables are required to be integer) where a formulation with an exponential number of variables may have been the most reasonable and perhaps only way to model a problem. Initially, column generation for integer programming served primarily as a primal heuristic in which columns were only generated at the root node, and conventional branching techniques and rounding heuristics were used on the generated columns to produce integer solutions (see for example Appelgren [4, 5]). The introduction of specialized branching techniques by Ryan and Foster [96] meant that one could now drive out fractional solutions without having to branch on column variables. Thus, preserving the structure and complexity of the pricing problem throughout the branch and bound process and transforming the use of column generation from a heuristic to an exact method for 0-1 integer programming. The early work on the crew scheduling and vehicle routing problems best exemplifies the state of the art in column generation during this period. See for example Ryan and Falkner [95], Desrochers and Soumis [44, 43], Agarwal et al. [2], and Desrosiers et al. [46]. In addition to the branching techniques, the development of efficient DP algorithms related to solving constrained shortest path problems for the resulting pricing

problem meant that a richer, more complex class of problems could now be solved that would be otherwise impossible to do explicitly within an integer program.

It wasn't until the 90s that the full extent of the connections between Dantzig-Wolfe decomposition, Lagrangean relaxation, and integer programming were being unraveled by researchers, sparking a renewed interest in column generation and the development of a unified framework for Dantzig-Wolfe decomposition and integer programming known as branch-and-price (see Barnhart et al. [11], and Desaulniers et al. [40] for an overview of branch-and-price including connections to Lagrangean relaxation, and Vanderbeck and Savelsbergh [113] for more recent developments related to mixed integer column generation reformulations). With improving computing power, it was now possible to use column generation to strengthen formulations, reduce symmetry compared to more compact formulations, and solve problems when traditional branch-and-cut algorithms struggled. This progress prompted a flood of research activity related to using column generation, in particular using branch-and-price, to solve many large scale problems in industry. The work during the early part of this decade on airline crew scheduling and vehicle routing (see for example Anbil et al. [3], Desaulniers et al. [41], Vance [109], and Ribeiro and Soumis [90]), paved the way for column generation being recognized as an invaluable tool in mainstream operations research.

Today, branch-and-price algorithms often coupled with cut generation (branch-price-and-cut algorithms) have become some of the most sophisticated and successful algorithms for solving several classes of very difficult and large scale discrete optimization problems. Examples include the generalized assignment problem (Savelsbergh [98]), binary cutting stock (Vance et al. [110] and Carvalho [106]), bin packing (Carvalho [107]), vehicle routing and variants (Fukasawa et al. [61], Cordeau et al. [31], and Ropke and Cordeau [94]), crew scheduling (Gamache et al. [62] and Day and Ryan [38]), network design (Barnhart et al. [10] and Parker and Ryan [88]), graph

coloring (Mehrotra and Trick [86]), fleet assignment (Barnhart et al. [9]), machine scheduling (Chen and Powell [24], and Van den Akker et al. [108]), etc. Together with developments in dual variable selection and stabilization techniques aimed at speeding up the convergence of the column generation LP (e.g. interior point methods (Elhedhli and Goffin [52]), bundle methods (Lemarèchal et al. [80]), the volume algorithm (Barahona and Anbil [7]), dual stabilization (Ben Amor et al. [16]), and constraint aggregation (Elhallaoui et al. [51])), developments in branching techniques (Vanderbeck [111]), and polyhedral developments related to "robust" cut generation (Poggi De Aragão and Uchoa [89]), the success is due in a large part to the development of efficient routines for solving the various pricing problems. Of special interest is the development of efficient DP algorithms for the *resource constrained shortest path problem* (RCSPP), a shortest path problem with values assigned to individual nodes and/or arcs corresponding to the consumption of certain resources, and constraints limiting the total consumption of these resources along the chosen path. RCSPP appears as part of substructures of many important classes of problems, including almost all the examples given thus far. It is therefore not surprising that it has become almost synonymous with column generation and branch-and-price prompting significant research aimed at improving techniques to solve it.

1.1 Thesis Outline and Contributions

Like Ford and Fulkerson half a century ago, we continue to strive for ever improving techniques to solve even larger, even harder problems than before. In terms of branch-price-and-cut, two obstacles we face today are being able to solve harder and larger pricing problems, and solving mixed-integer column generation formulations that suffer from relatively weak LP bounds compared to the more traditional 0-1 set partitioning type. As part of the work presented in this thesis, we encounter column generation formulations motivated by real life problems that require overcoming both

types of challenges.

In Chapter 2 we address RCSPP on extremely large networks with many resource constraints. Although RCSPP itself has received considerable attention in the context of column generation, the size of pricing problems that can be solved efficiently is often the bottleneck to solving larger problems. This is particularly a problem when time-expanded networks are used to model a problem. An example is the recently proposed formulation for the Dial-a-Flight Problem (DAFP) (Espinoza et al. [54]). DAFP arises in the context of a per-seat on-demand air transportation service that operates without a fixed schedule. In DAFP, the routing of jets and the assignment of passengers to jets need to be considered simultaneously. As a result, the time-activity networks and the associated set of resource constraints needed to represent possible jet and passenger itineraries grow rapidly with the number of airports and requests for transportation. Although in this case the column generation formulation resembles a traditional 0-1 type formulation, the sheer size of the networks and number of resource constraints (in the order of millions of nodes and arcs, and thousands of resource constraints) makes the resulting pricing problem intractable using standard DP techniques for RCSPP. We develop a relaxation based DP algorithm for RCSPP that exploits the local structure of resource constraints typically found in time-expanded formulations, and that alternates between a forward and a backward search. Each search employs bounds derived in the previous search to prune the search space. Between consecutive searches, the relaxation is tightened using a set of critical resources and a set of critical arcs over which these resources are consumed. As a result, a relatively small state space is maintained and many paths can be pruned while guaranteeing that an optimal path is ultimately found. In the grander scheme, the success resulting from solving larger pricing problems translates to proving near optimal bounds for instances of DAFP that are otherwise impossible to obtain.

In Chapter 3 we address the *fixed charge shortest path problem* (FCSPP) which is

a variant of RCSPP in which the amount of resource consumed when traversing an arc/node is itself a continuous bounded variable with linear cost. FCSPP appears as part of the pricing process of column generation formulations for several important classes of scheduling, transportation, and resource allocation problems in which the pricing problem couples together discrete decisions with knapsack type constraints on continuous variables. Examples include the split delivery vehicle routing problem (Dror et al. [47], Gendreau et al. [65], and Desaulniers [39]), the parallel machine scheduling problem with controllable processing times (Nowicki and Zdrzalka [87], Cheng et al. [25], and Daniels and Sarin [36]), inventory routing problems (Federgruen and Zipkin [56], Campbell et al. [21], and Christiansen [26]), and production planning problems with flexible specifications (Balakrishnan and Geunes [6]). Column generation formulations have proved very successful for the more traditional variants of these examples in which resource consumption along nodes/arcs is fixed and the pricing problem is an RCSPP. Unfortunately, the perceived complexity of having resource consumption itself being a continuous variable, has deterred any such use for the more practical variants. We develop a new dynamic programming algorithm for FCSPP that uses solutions from labeling and dominance techniques for standard RCSPP on slightly modified problems, and combines these solutions by exploiting the structure provided by certain classes of knapsack problems to efficiently construct an optimal solution to FCSPP. We are thus able to offer efficient solution methods for new, more practical variants of problems that are otherwise difficult to solve using standard integer programming techniques, and otherwise restricted to heuristic search and approximation algorithms.

Finally, shifting our focus away from just the pricing problem, in Chapter 4 we investigate the use of branch-price-and-cut as a solution method to solve mixed 0-1 programs related to maritime inventory routing. The resulting mixed 0-1 column generation reformulation has several peculiarities and poses several new challenges

compared to pure 0-1 reformulations. In addition to the fact that the pricing problem involves solving an FCSPP that includes both continuous and 0-1 variables, integrality of the reformulation is required on auxiliary variables rather than (as is more common with pure 0-1 reformulations) on the column generation variables. Secondly, although an improvement over compact formulations, these mixed 0-1 reformulations generally have much weaker bounds than pure 0-1 reformulations and thus require significant effort to further tighten the formulation through pre-processing and cut generation. Within a branch-price-and-cut framework, this also requires additional care to ensure that the structure and complexity of the pricing problem remains tractable. Here we use the DP developed for FCSPP to solve the pricing problem. To further tighten the formulation, we use the problem's boundary conditions during preprocessing and to restrict the set of columns that are produced by the pricing problem. We introduce branching schemes and cuts that can be implemented efficiently and that preserve the structure of the pricing problem. Some of the cuts are inspired by the capacity cuts known for the vehicle routing problem. Although these cuts are useful to tighten the relaxation, they are somewhat limited since they are derived from purely binary implications. We also derive a new class of mixed 0-1 cuts that considers both binary and continuous variables specifically targeting fractional solutions brought about by individual vessels "competing" for limited inventory at load ports and limited storage capacity at discharge ports. In all, we believe the proposed branch-price-and-cut approach is unique for inventory routing problems containing several innovations including the pricing and the cuts. We can solve practically sized problems to optimality and produce reasonable bounds for harder instances when alternative branch-and-cut approaches fail.

In summary, the work presented in this thesis makes several contributions including

1. the development of efficient DP techniques to solve RCSPP in particular, efficient techniques for RCSPP in large time-expanded networks with many resource constraints,

2. the development of a new DP algorithm to solve FCSPP, a difficult generalization of RCSPP that has not been addressed before although it appears in several important classes of problems, and

3. a branch-price-and-cut algorithm for the inventory routing problem, an example of a difficult mixed 0-1 problem that suffers from weak LP bounds and that requires considerable effort in preprocessing, branching, and cut generation outside the realms of branch-price-and-cut for traditional 0-1 problems.

In the remainder of this chapter, we give a brief technical overview of branch-price-and-cut to provide the reader with some general background on the subject, and set the tone for terminology that will be used throughout the thesis as well as highlight the challenges that lay before us.

1.2 Branch-Price-and-Cut: An Overview

Branch-price-and-cut is a framework for solving mixed integer problems with a prohibitively large number of variables. Such formulations are typically obtained as a result of applying the Dantzig-Wolfe decomposition principle ([37]) on a compact formulation. The decomposition scheme is particularly applicable for compact formulations whose constraints admit a natural decomposition into subproblems that can be solved much more efficiently compared to solving the compact formulation in its entirety.

Consider the mixed integer problem (MIP)

$$\begin{aligned} \min\ & c(x,z) \\ \text{s.t. } & A(x,z) \leq\ a \\ & B(x,z) \leq\ b \\ & x \in \Re_+^{n_1}, z \in \mathbb{Z}_+^{n_2} \end{aligned}$$

where A and B are $m_1 \times (n_1 + n_2)$ and $m_2 \times (n_1 + n_2)$ rational matrices respectively. If the system $B(x,z) \leq b$ corresponds to a more tractable problem than the system coupled with $A(x,z) \leq a$, and/or has a special structure, for example, if B has a block diagonal structure

$$B = \begin{bmatrix} B^1 & & & \\ & B^2 & & \\ & & \ddots & \\ & & & B^K \end{bmatrix}$$

where B^i is a $m_2^i \times (n_1^i + n_2^i)$ sub-matrix of B and b^i is the corresponding right hand side, then the above problem is an ideal candidate for reformulation and solving by branch-price-and-cut. Let (x_q^i, z_q^i) $q = 1, \ldots, Q^i$ be the indexed set of extreme points of the convex hull of

$$P^i = \{(x \in \Re^{n_1^i}, z \in \mathbb{Z}^{n_2^i}) : B^i(x,z) \leq b^i)\}.$$

Assuming P^i is bounded, the Dantzig-Wolfe reformulation (DW) of MIP over B can

then be stated as

$$\begin{aligned}
\min \sum_{i=1,\ldots,K} \sum_{q=1,\ldots,Q^i} & c^i(x^i_q, z^i_q)\lambda^i_q & & \\
\text{s.t.} \sum_{i=1,\ldots,K} \sum_{q=1,\ldots,Q^i} & A^i(x^i_q, z^i_q)\lambda^i_q \leq \ a & & (1.1)\\
\sum_{q=1,\ldots,Q^i} & z^i_q \lambda^i_q \in \mathbb{Z}^{n_2^i}_+ & \forall i = 1,\ldots,K & (1.2)\\
\sum_{q=1,\ldots,Q^i} & \lambda^i_q = 1 & \forall i = 1,\ldots,K & (1.3)\\
& \lambda^i_q \geq 0 & \forall i = 1,\ldots,K, \forall q = 1,\ldots,Q^i &
\end{aligned}$$

where $c = [c^1, \ldots, c^K]$ and $A = [A^1, \ldots, A^K]$ are the submatrices corresponding to the respective blocks. By ignoring the integrality requirements (1.2) above, we arrive at the linear relaxation of DW also known as the column generation master problem. To solve the master problem, negative reduced cost columns are added dynamically based on the dual values $\pi \in \Re^{m_1}$ corresponding to constraints (1.1), and α^i corresponding to constraints (1.3). As part of this pricing process, we need to solve for each block $i = 1, \ldots, K$

$$-\alpha^i + \arg\min\{(c^i - \pi A^i)(x, y) : x \in \Re^{n_1^i}, z \in \mathbb{Z}^{n_2^i} \text{ and } B^i(x, y) \leq b^i\}.$$

When integrality in DW is enforced using branch-and-bound in which the master problem is solved by column generation at each node of the branch-and-bound tree, the resulting framework is called branch-and-price.

There are several advantages to solving DW compared to its original compact counterpart MIP. Apart from the possibility of exploiting the decomposable structure and solving the individual blocks more efficiently, the bound obtained from the linear relaxation of DW is no worse, and typically much better than the linear relaxation of MIP when the linear relaxation of the block systems defining P^i do not have integer extreme points. Furthermore, if the K blocks are identical, the convexity

constraints (1.3) can be aggregated and only a single pricing problem is required. Perhaps more importantly, this eliminates solutions that are simply permutations of the index chosen for each block, therefore, removing symmetry that can be otherwise detrimental to the branch-and-bound process.

1.2.1 Solving the pricing problem

In the context of integer programming, the key is to ensure that any strengthening through Dantzig-Wolfe reformulation outweighs the effort required to solve the pricing problems. Since the pricing problem is more often than not an RCSPP, research related to solving the pricing problem is mostly focused on improving techniques for RCSPP.

DP based labeling algorithms are perhaps the most efficient of all approaches for RCSPP. In a labeling algorithm for RCSPP, a partial path is characterized by a label which contains a set of numerical attributes that can be used to assess its quality and extensibility relative to other paths (Desrosiers et al. [45] and Irnich and Desaulniers [77]). The set of all possible attribute values associated with feasible paths is called the state space of the problem. Assuming the attribute values are integer, maintaining only some efficient set of non-dominated labels typically leads to a pseudo-polynomial algorithm for RCSPP. We refer to Desaulniers et al. [77] for details and relative merits of various DP algorithms for RCSPP.

Although the origins of DP dates back to the early work of Bellman [14, 15], considerable improvements have been made with respect to the efficiency of these algorithms in practice. However the pseudo-polynomial complexity of these problems means that the worst case runtime of some of the more sophisticated DP algorithms today is the same as the simple recursive/induction algorithms obtained from Bellman's equations. That said, one has to be careful when drawing any conclusions

about the applicability of DP algorithms based on worst case analysis since the various acceleration schemes embedded in DP algorithms today make them much more efficient in practice than their worst case run time and or state space suggest. We will discuss some of these acceleration techniques in the context of solving RCSPP in Chapter 2.

1.2.2 Enforcing integrality

If the original formulation, and thus also the reformulation is a pure 0-1 program, then the resulting integrality and convexity constraints of DW (constraints (1.2) and (1.3)) boil down to enforcing $\lambda_q^i \in \{0, 1\}$, i.e., enforcing integrality on the column generation variables. A conventional branching scheme can be used in which we set λ_q^i to 0 in one branch, and to 1 in the other. However, this type of dichotomy leads to a very imbalanced branch-and-bound tree and perhaps more troubling, requires us to significantly change the structure of the pricing problem. Indeed, the pricing problem for a node in the resulting branch-and-bound tree requires us to find an entering column that does not correspond to any of the columns previously fixed to 0, potentially requiring the k^{th} best solution to the pricing problem where k is the depth of the node in the branch-and-bound tree. This results in a potentially much harder pricing problem than originally intended. To sidestep this problem, one may be tempted to include the branching decision as an additional constraint in the master problem. The difficulty here however is in incorporating the dual values of these additional constraints into the pricing problem in such a way that they only appear in the reduced cost calculation of the appropriate columns that we are trying to fix to 0. Invariably, this is the just as hard as finding the k^{th} best solution to the pricing problem landing us back where we started.

Rather than branching on the column generation variables, Ryan and Foster [96]

recognized in the context of solving 0-1 set partitioning problems that in any fractional solution there always exists two rows such that the sum of fractional values of columns covering both rows is also fractional. Based on this observation, they proposed a branching scheme in which columns covering both rows are set to 0 in one branch, and set to 1 in the other. Such a branching scheme leads not only to a more balanced branch-and-bound tree but within a branch-and-price framework, the difficulty involved with generating columns that either covers both rows or at most one of the rows is generally much easier than having to find the k^{th} best solution. Moreover, if one of the two rows is also a convexity constraint corresponding to a particular block (i.e., constraint (1.3) in DW), then the above branching scheme results in either forcing a particular block to cover a row or not cover a row. For the 0-1 case, this form of "constraint branching" is analogous to fixing a binary variable of the compact formulation which in most cases can be done trivially without having to change the structure of the pricing problem.

In the case that the pricing problem is an RCSPP, the variables of the compact formulation are simply the binary variables associated with traversing an arc and/or visiting a node. For the above constraint branching scheme, the 0 branch can then be trivially enforced by simply removing the node or arc, and the 1 branch can be enforced by modifying the network structure in most cases or at the very least, by modifying the costs so as to encourage the optimal solution to use the required node/arc. In Chapter 4 we show how this can be done for a reformulation of the inventory routing problem that contains both binary and continuous variables. We refer to Barnhart et al. [11] for details and references therein for other applications of this branching scheme.

As demonstrated by Barnhart et al., enforcing integrality for reformulations of general mixed integer problems can also be carried out through single variable disjunctions in the space of the compact formulation. If the solution method for solving

the pricing problem is independent of the variable bounds chosen as a result of these disjunctions, then the structure of the pricing problem remains unchanged. In the case that the block structures are identical, the branching scheme can typically be improved to alleviate some of the symmetry in the problem at the cost of adding additional constraints in the master problem and/or slightly altering the structure of the pricing problem. We refer again to Barnhart et al. [11], Vanderbeck and Savelsbergh [113], and Vanderbeck [111, 112] for details of the proposed branching schemes, their relative merits, and applications.

1.2.3 Cut generation

To strengthen MIP, one can add inequalities known as cuts that do not remove any feasible integer solutions but that violate certain fractional solutions to its linear relaxation. A branch-price-and-cut framework results from adding cuts to further strengthen DW during the branch-and-price process. The challenges we face in adding cuts to DW are analogous to the challenges one faces when making a choice of branching decisions. Incorporating the dual values corresponding to a cut into the pricing problem may prove to be difficult if the cut cannot be stated in the space of the compact formulation. If a cut can be stated in the space of the compact formulation, then the corresponding dual value can easily be incorporated into the pricing problem by appropriately modifying the costs associated with the variables in the pricing problem. Although some were likely aware through the work on branching strategies for branch-and-price that cuts could be added without disrupting the structure of the pricing problem, it was probably perceived as being too complex an integration for any further improvement in bound that may result. The successful implementation and application of a branch-price-and-cut framework by Fukasawa et al. [61] for the vehicle routing problem forced many to rethink this notion of branch-and-price and branch-and-cut as being two separate and competing entities. Fukasawa et al. used

the term "robust" to characterize cuts that can be added without having to further complicate the pricing problem. In Chapter 4 we use several classes of robust cuts to strengthen our formulation for the inventory routing problem. Indeed, without these cuts it would be practically impossible to solve even small problems. We refer to Poggi De Aragõ and Uchoa [89] for a more in depth discussion related to this topic of robust cuts including several other successful examples of its application.

A disadvantage of only considering robust cuts that can be stated in the space of the compact formulation is that many of these cuts may be implied by the column generation formulation. Any cut that is valid for individual blocks P^i, $i = 1, \ldots, K$ is obviously implied by the column generation formulation. In some cases, cuts that are valid for multiple blocks at a time may also be implied, although this is generally less trivial to prove. The analysis provided by Letchford and Salazar-González [81] and Ropke and Cordeau [94] for the vehicle routing and pickup and delivery problems respectively are good examples of analysis on the implication of robust cuts by the column generation formulation. In these examples, although there exists a myriad of cuts available for the compact formulation, only a handful of these are shown not to be implied by the column generation formulation leaving one to often look elsewhere for cuts to further strengthen the formulation.

By only considering robust cuts, we may ignore a rich source of structural cuts that can be applied in terms of the column generation variables. Clique inequalities for set partitioning for example may prove very useful for strengthening 0-1 type column generation formulations, although this may add additional complexity to the pricing problem. Spoorendonk and Desaulniers [104] have successfully used clique inequalities as part of a branch-price-and-cut algorithm for the vehicle routing problem with time windows. Here, for each clique inequality that is added to the master problem, an additional resource constraint is added to the pricing problem, an RC-SPP. Although these cuts are not robust, the fact that the pricing problem can still

be solved relatively efficiently (because of the improving efficiency of DP techniques for RCSPP) means that the gap can be closed even further without much overhead. We refer to Spoorendonk's thesis ([103]) for examples of other non-robust cuts that have been successfully applied within branch-and-price.

1.2.4 Connection to Lagrangean relaxation

As an alternative to the column generation master problem, another relaxation of MIP can be obtained through Lagrangean relaxation when dualizing the constraints defined by A and penalizing this relaxation by $\pi \in \Re_+^{m_1}$. The resulting Lagrangean dual problem is

$$\begin{aligned} \max_{\pi \geq 0} \quad & \min \ c(x,z) + \pi(b - A(x,z)) \\ & \text{s.t. } B(x,z) \leq \ b \\ & x \in \Re_+^{n_1}, z \in \mathbb{Z}_+^{n_2}. \end{aligned}$$

Note that after replacing (x, z) with a convex combination of (x_q^i, z_q^i) for each block $i = 1, \ldots, K$ and $q = 1, \ldots, Q^i$, and then taking the dual of the resulting minimization component, we arrive at the dual of the linear relaxation of DW (i.e., the dual of the master problem) (Geoffrion et al. [66]). Thus, the bounds produced by the master problem and the above Lagrangean relaxation are equivalent. Moreover for a given π, the resulting problem is the same as the column generation pricing problem and is used as a subproblem to evaluate the quality of the Lagrange multipliers during an update step. To solve the Lagrangean dual, one can take advantage of the abundant number of sophisticated subgradient (Kallehauge et al. [79]), cutting plane (Goffin and Vial [70]), and bundle (Lemarèchal et al. [80]) methods that produce good quality dual multipliers relatively quickly (see Briant et al. [20] for computational comparisons). Despite this, the column generation relaxation seems to be the preferred route for most since branching and adding cuts are readily adoptable to a relaxation that maintains primal rather than dual feasibility, and since dual stabilization techniques

(du Merle et al. [48]) can be embedded within column generation to mimic bundle type convergence and alleviate the tailing off behavior. We refer to Huisman et al. [76] for further details on the connections to Lagrangean relaxation.

CHAPTER II

SHORTEST PATH BASED COLUMN GENERATION ON LARGE NETWORKS WITH MANY RESOURCE CONSTRAINTS

2.1 Introduction

As part of the column generation pricing process, one often needs to solve a *resource constrained shortest path problem* (RCSPP). We consider solving RCSPP as part of a column generation pricing process for formulations involving extremely large networks and a huge number of resource constraints. In practice, such formulations typically occur when time-expanded networks are used to model a problem and when resources are consumed and replenished over time as certain tasks are completed and new tasks undertaken.

An example of a large time-expanded formulation is the recently proposed formulation for the Dial-a-Flight Problem (DAFP) (Espinoza et al. [54]). DAFP arises in the context of a per-seat on-demand air transportation service that operates without a fixed schedule. In DAFP, the routing of jets and the assignment of passengers to jets need to be considered simultaneously. The time-activity networks and the associated set of resource constraints needed to represent possible jet and passenger itineraries grow rapidly with the number of airports and requests for transportation. Column generation using standard dynamic programming (DP) techniques for RCSPP becomes intractable even with acyclic networks, tight dominance, and moderately sized instances.

In this chapter, we present a novel DP-based search procedure that contains the following innovations:

- A dominance scheme that exploits the fact that a resource may accumulate only locally within the network.

- An iterative scheme that alternates between a forward and backward search, which gives rise to a natural bounding scheme for pruning the search, and that employs a dynamically changing and progressively tighter relaxation to control the size of the state space while ensuring that an optimal feasible path is ultimately found.

- A unique form of pre-processing that makes use of the structure typically encountered in time-expanded networks.

A computational study using instances of DAFP and the formulation proposed in Espinoza et al. [54] demonstrates the merits of the proposed approach. Instances with up to 200 jets and 1600 requests for transportation are considered resulting in network representations in excess of millions of nodes, arcs, and resource constraints. The size of these networks and the number of resource constraints is an order of magnitude larger than those that even the most sophisticated RCSPP algorithms reported in the literature can handle. The larger DAFP instances we consider are more than 20 times the size of Dial-a-Ride Problem instances with comparable vehicle capacities considered in the literature. Solution methods for these problems have been limited to branch-and-cut algorithms or using column generation only as a heuristic (see Cordeau [30] and Cordeau and Laporte [32]). Furthermore, although one has to be cautious when making comparisons to instances of the pickup-and-delivery problem, it is still of significance that the sizes of instances we consider are much larger than the most recent work in this area as well. Ropke and Cordeau [94] and Xu et al. [114] specifically comment on the intractability of the pricing problem when there are more than a few hundred requests for transportation.

In the remainder of this chapter, we describe relevant literature in Section 2.2,

formally define the problem of interest in Section 2.3, discuss the proposed solution approach in Section 2.4, and present an extensive computational study in Section 2.5.

2.2 Literature Review

It is well-known that RCSPP is NP-hard even with acyclic networks, positive costs and a single resource constraint (see Garey and Johnson [64]). Nonetheless, one can find an abundance of algorithms for RCSPP. The most common include approximation algorithms (Hassin [74] and Lorenz and Raz [82]), Lagrangean based methods (Beasley and Christofides [12], Handler and Zhang [73], and Mehlhorn and Ziegelmann [85]), polyhedral approaches (Spoorendonk and Petersen [105], and Garcia [63]), and DP algorithms (as in Desrosiers et al. [45] for modeling capacitated and time constrained routing, and in Irnich and Desaulniers [77] for general constrained shortest path problems (CSPP) dealing with various types of side constraints).

Although an interesting problem in itself, RCSPP and its variants usually occur as part of a column generation formulation of many important classes of problems, including the vehicle routing problem (VRP), the vehicle routing problem with time-windows (VRPTW) and pickup-and-delivery problem with time windows (PDPTW), the fleet assignment model, crew-pairing and rostering, train scheduling and line planning, maritime inventory routing, and network bandwidth design to name just a few. See Cordeau et al. [34] and Cordeau et al. [33] for an exposition of formulations and solution methods (including branch-price-and-cut) for various types of vehicle routing and pickup-and-delivery problems, Christiansen et al. [27] for maritime routing and inventory management problems, and Borndörfer et al. [19], Cordeau et al. [35], Haase et al. [72], Holmberg and Yuan [75], and Sandhu and Klabjan [97] for recent examples of CSPP-based column generation formulations and solution methods for complex integrated models in passenger transportation, scheduling, and network design.

Although, the core components for solving CSPP by DP have remained mostly unchanged since the seminal work of Desrochers [42], advances in DP techniques in two areas have led to significant improvements. Firstly, improvements in strengthening the sufficiency conditions used for proving dominance of one partial path over another by exploiting one's specific knowledge of the problem at hand. Examples of this are Feillet et al. [57] and Ropke and Cordeau [94] on the resulting pricing problem of VRPTW and PDPTW, respectively. Another area of improvement has been augmenting the basic algorithm using various preprocessing, scaling, bounding, and search strategies. For example, Dumitrescu and Boland [50] focus on pre-processing and label-elimination methods through resource and cost-based bounding schemes in acyclic networks and a single resource constraint. They use shortest path algorithms to compute the minimum amount of resource that can be accumulated for a path from the source to all other nodes and from all other nodes to the sink. This information is used in a preprocessing phase to remove from the network nodes and arcs that can be proven not to be part of any feasible path as well as during the DP algorithm to eliminate labels that can be proven not to be extendable to the sink with feasible resource consumption. An analogous scheme is outlined for preprocessing and pruning by cost rather than by resource consumption. They show that notable improvements to pruning are possible by tightening the cost bounds through Lagrangean based methods. Lübbecke [83] also uses cost-based bounds to prune the search. The bounds are computed not just for the completion of a path terminating at a node, but for a specific label. A bound on the best possible extension of a label is computed by greedily including the dual values obtained from the master problem in the cost of completing the label. Lübbecke also points out that one can identify a priori certain conflicts (for example caused by resource consumption) to strengthen this bound while continuing to use a greedy approach. Another use of bounding is the bi-directional DP outlined in Righini and Salani [92], where two concurrent searches

lead to labels being pruned at some "half-way" point if compatible labels in the opposite direction cannot be found to create a path from source to sink. The algorithm was shown to have a significant impact on the efficiency of pricing for VRP. Finally, Feillet et al. [58] outline acceleration strategies that exploit information from the current solution of the column generation relaxation to "hot-start" the DP.

Recently, the drive for tighter linear relaxations by enforcing certain structural constraints, such as path elementarity in the pricing problems for VRP, VRPTW, and PDPTW, make solving RCSPP of otherwise benign pricing problems extremely difficult. The resulting pricing problem in such cases is NP-hard in the strong sense. As pointed out in Beasley and Christofides [12], enforcing path elementarity can be viewed as enforcing additional resource constraints, generally one for each node in the network, each limiting the number of visits to a node. Thus, standard DP techniques can be applied when solving RCSPP with path elementarity. However, as demonstrated by Feillet et al. [57], the size of the state space explored by such an algorithm becomes unmanageable even for relatively small networks. Although the strengthening of the dominance scheme as proposed by Feillet et al. [57] goes some way to alleviate this burden, their computational tests show that the growth of the state space when the number of nodes increases cannot simply be overcome by strengthening the dominance scheme or through label-elimination procedures alone. Research in this area has pointed to relaxation-based DP schemes in which path elementarity is enforced only on a subset of nodes, which is iteratively updated to ensure the optimal elementary path is ultimately found. Righini and Salani [91] call this a "decremental" state space relaxation and implement this within a bi-directional DP. Boland et al. [18] propose an analogous scheme, developed independently, using a strengthening of dominance similar to that proposed by Feillet et al. [57] rather than using a bi-directional search to eliminate labels.

2.3 Problem Description and Notation

In this section, we formally introduce RCSPP together with notation that will be used throughout the chapter. The set of tasks to be performed is denoted by $\mathcal{R}$. Examples of a task include executing a scheduled flight (crew pairing), satisfying the demand at a certain location within a specified time window (vehicle routing), and transporting a passenger from a given origin to a given destination respecting a limit on travel duration (dial-a-ride). The set of commodities available to perform these tasks is denoted by $\mathcal{J}$. For routing problems commodities typically correspond to vehicles or jets. For crew-pairing commodities correspond to the actual crews.

We denote by $\mathcal{N}_j = (N_j, A_j)$ the network for each $j \in \mathcal{J}$, i.e., N_j is the node set and A_j is the arc set, and define $n_j^s \in N_j$ and $n_j^t \in N_j$ to be the source and sink nodes, respectively. A path from source to sink in such a network represents a sequence of activities that can be carried out by the commodity in order to perform tasks. For example, in the VRP a path corresponds to a route for a vehicle satisfying one or more demands, in the DAFP a path corresponds to jet and passenger itineraries, and in the crew-pairing problem a path corresponds to a tour-of-duty for a crew member. However, not every path from source to sink is feasible because of resource consumption. For example, in the natural network representation of the VRP, where a node corresponds to the depot or the demand at a certain location and an arc to traveling between two locations, not every path is feasible since the vehicle capacity may be exceeded. Therefore, to maintain feasibility, the only resource that needs to be tracked is capacity. With the addition of time windows, one also needs to track time, which "accumulates" along the route. For other problems, such as DAFP and the crew-pairing problem, several resources need to be tracked to be able to ensure feasibility of the path from source to sink. The networks proposed in Espinoza et al. [54] to model DAFP, for example, require resource constraints for each flight leg to ensure jet capacity and weight limits are never exceeded. Additional resource constraints are

imposed to limit the flying-time during a shift and to ensure a transportation request is never picked-up more than once.

To account for the cost of a path, for the tasks it covers, and for the resources it consumes, we define for each arc $e \in A_j$, a cost c_e, non-negative integer weights g_e^k corresponding to the amount of resource k being consumed ($k \in \{1, \ldots, K\}$), and binary indicators h_e^r specifying whether task r is performed when traversing arc e ($r \in \mathcal{R}$). Given a path P, we use $A(P)$ to denote the sequence of arcs traversed in P, $c(P) = \sum_{e \in A(P)} c_e$ to denote the cost associated with P, $h^r(P) = \sum_{e \in A(P)} h_e^r$ to denote the number of times task r is performed, and $g^k(P) = \sum_{e \in A(P)} g_e^k$ to denote the amount of resource k that is accumulated along P.

The primary objective of a column generation formulation is to find a minimum cost set of paths from source to sink for each $j \in \mathcal{J}$ (generally one for each commodity), so that all tasks are completed and the amount of resource $k \in \{1, \ldots, K\}$ consumed along a path does not exceed the amount available, denoted by the integer b^k. Thus, as part of the column generation pricing process, we are required to solve for each commodity $j \in \mathcal{J}$, the pricing problem:

$$\min \bar{c}(P) = \sum_{e \in A(P)} \left(c_e - \sum_{r \in \mathcal{R}} \pi_r h_e^r \right) - \alpha_j$$

$$\text{s.t. } P \text{ is a path from } n_j^s \text{ to } n_j^t \text{ in } \mathcal{N}_j, \text{ and}$$

$$g^k(P) \leq b^k \text{ for } k = 1, \ldots, K.$$

The dual values π_r for each task $r \in \mathcal{R}$ and α_j for each commodity $j \in \mathcal{J}$ are obtained

from the solution to the following restricted master problem (RMP):

$$\begin{aligned}
\min \sum_{j\in\mathcal{J}}\sum_{P\in\Psi_j} c(P)\lambda_P & \\
\text{s.t.} \sum_{j\in\mathcal{J}}\sum_{P\in\Psi_j} h^r(P)\lambda_P = 1 & \quad \forall r \in \mathcal{R} \\
\sum_{P\in\Psi_j} \lambda_P = 1 & \quad \forall j \in \mathcal{J} \\
\lambda_P \geq 0 & \quad \forall P \in \Psi_j \text{ and } \forall j \in \mathcal{J}.
\end{aligned}$$

Here Ψ_j represents some subset of all (resource) feasible paths between source and sink for commodity j, including the ones generated by the pricing problem so far. By embedding the dual costs within the arcs, we do not need to make a distinction when referring to min-cost and min-reduced-cost paths and therefore we use these terms interchangeably throughout the chapter. Moreover, we assume that if path elementarity is required, we enforce this through resource constraints provided as part of the problem description.

2.4 Solution Approach

In this section, we present a DP-based search procedure that incorporates a number of complementary techniques for efficiently and effectively managing the size and exploration of the state space and that allows the solution of RCSPP on extremely large networks with a huge number of resource constraints.

2.4.1 Conditions for Path Feasibility and Dominance

Definition 2.4.1. Let P be a path in $\mathcal{N}_j$. P is feasible if $g^k(P) \leq b^k$ for $k = 1, \ldots, K$.

Definition 2.4.2. Given two feasible paths P_1 and P_2 from the source to node $n \in N_j$, we say that P_1 dominates P_2, denoted $P_1 \preceq P_2$, if:

1. $c(P_1) \leq c(P_2)$, and

2. any feasible extensions of P_2 by a path from n to the sink is also a feasible extension for P_1.

DP algorithms for RCSPP discard a path P_2 if there exists an alternative path P_1 that starts and ends at the same nodes and satisfies $c(P_1) \leq c(P_2)$ and $g^k(P_1) \leq g^k(P_2)$ for all $k \in \{1, \ldots, K\}$, since in this case $P_1 \preceq P_2$. However, using this sufficient condition for dominance alone may lead to exploring an unnecessarily large portion of the state space. Indeed, if a given resource accumulates only locally within the network, as is typically the case with time-expanded formulations, then it might be possible to identify regions within the network, such that an extension of a feasible path outside this region is guaranteed not to lead to infeasibility with respect to the given resource. In this case, one need not check dominance with respect to this resource if extending paths outside this region, which may allow elimination of paths that would otherwise not be eliminated. To capture these ideas more formally, we introduce the notion of *support* for a resource constraint.

Definition 2.4.3. For each $j \in \mathcal{J}$ and $k = 1, \ldots, K$, we say that $N_j^k \subseteq N_j$ is a support of k in network $\mathcal{N}_j$ if it contains all nodes $n \in N_j$ for which there exists a path P_1 from the source to n and a path P_2 from n to the sink with:

1. $g^k(P_1) + g^k(P_2) > b^k$,
2. $g^k(P_1) > 0$, and
3. $g^k(P_2) > 0$.

Thus, if $n \in N_j \setminus N_j^k$, i.e., n is not in the support of k in network $\mathcal{N}_j$, then for each path P_1 from the source to n and each path P_2 from n to sink we have that either:

1. $g^k(P_1) + g^k(P_2) \leq b^k$,
2. $g^k(P_1) = 0$, or

3. $g^k(P_2) = 0$.

We use the term *local resource constraints* when referring to resource constraints whose support is considerably smaller than the number of nodes in the network. Given a support for each resource, we can modify the criteria for dominance by only considering a resource if the paths being compared end at a node in the support for that resource.

Proposition 2.4.4. *Given two feasible paths P_1 and P_2 from the source to node $n \in N_j$, $P_1 \preceq P_2$ whenever:*

1. $c(P_1) \leq c(P_2)$ *and*

2. $g^k(P_1) \leq g^k(P_2)$ *for each $k = 1, \ldots, K$ such that $n \in N_j^k$.*

A standard dynamic program, as outlined in Irnich and Desaulniers [77] for example, can then be used with the dominance criteria just proposed to solve RCSPP.

Since the maximum number of non-dominated feasible paths stored during the course of the DP algorithm when using the proposed dominance criteria depends on the number of supports containing n for each node $n \in N_j$, theoretically it is advantageous to compute the minimum support for each resource. We can compute the minimum support for a resource $k \in \{1, \ldots, K\}$ by computing the longest path with respect to the consumption of k from the source to each node and from each node to the sink. Moreover, if we compute this information, we can strengthen the dominance depending on the paths we are comparing. Suppose, for example, that we have two paths P_1 and P_2 from the source to n with $\max\{g^k(P_1), g^k(P_2)\} \leq \mu$. If the longest path (with respect to resource k) from n to the sink consumes no more than $b^k - \mu$ of resource k, then we can ignore checking dominance with respect to resource k when comparing paths P_1 and P_2, even if $n \in N_j^k$.

In many cases, however, it may be possible to find reasonably sized supports without having to solve auxiliary optimization problems that may be extremely time

consuming. In DAFP and the network representation proposed in Espinoza et al. [54], for example, one can immediately determine for each node $n \in N_j$ and resource $k \in \{1, \ldots, K\}$, whether there exists a path P_1 from source to n and a path P_2 from n to the sink with $g^k(P_1) > 0$ and $g^k(P_2) > 0$. If so, we include n in the support of k. As we are using a weaker condition on the necessity to include a node in the support as defined in Definition 2.4.3, we may not end up with a minimal support, but it may still be much smaller than N_j and it is determined without much computational effort.

As a final note, we point out that using supports to enhance dominance is not the same as using resource-based bounding schemes to eliminate labels. Rather than exploiting the fact that a path corresponding to a particular state may not be feasibly extendable to the sink, we instead exploit the fact that a path may always be feasibly extended to the sink. This subtle yet important difference is particularly crucial for networks that are judiciously constructed or in the presence of local resource constraints where feasible paths can almost always be feasibly extended to the sink. In such cases, resource-based bounding schemes as used for example in Dumitrescu and Boland [50] are likely to have little or no impact.

2.4.2 An Arc-Based Relaxation for RCSPP

The number of paths explored using a standard DP algorithm with the proposed dominance scheme grows with network size and number of resources, and is likely to far outpace any reasonable demand on computing power and memory, making RCSPP intractable using standard DP techniques even when exploiting the structure provided by local resource constraints. To explore fewer paths, we incorporate relaxation ideas used to decrease the state space for elementary shortest-path computation (see Righini and Salani [91] and Boland et al. [18]) and extend them to apply more generally within the context of RCSPP.

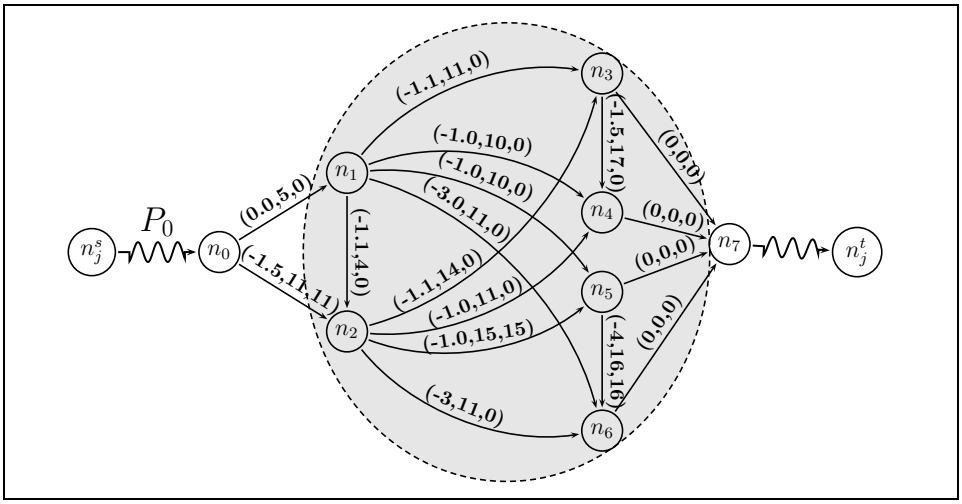


Figure 2.1: An example illustrating the reduction in state space when using an arc-based relaxation of resource consumption.

Rather than keeping track of resource consumption over all arcs, we can instead keep track of the consumption of a given resource only on a subset of arcs. Although this idea is related to scaling, a well-known technique used to project the state space of a DP onto a smaller state space, it is significantly different because the "scaling" is only performed for a subset of commodities and for each of these commodities only on a subset of arcs. Moreover, we use a very simple implementation of scaling in which for a given arc there is either no scaling of the resource consumption, or the resource consumption is ignored entirely. For a given resource, scaling is normally applied uniformly over all arcs (see for example Dumitrescu and Boland [50]). Although, the reduction in the size of the state space generally leads to a reduction in the number of paths explored, this generally comes at the expense of no longer being able to guarantee feasibility of the optimal path found. However, by judiciously choosing the resources and the arcs, we are able to reduce the size of the state space *and* ensure feasibility of the optimal path found. We next highlight the potential benefits from using an arc-based relaxation of resource consumption by demonstrating its impact on a simple example.

In Figure 2.1, we attempt to construct the optimal extension of path P_0 through the sub-network induced by the nodes in the support $N_j^k = \{n_1, n_2, \ldots, n_6\}$ of resource k. Alongside each arc e in Figure 2.1, we display three pieces of information: the cost c_e, the true amount g_e^k of resource k consumed when traversing the arc, and the amount $\tilde{g}_e^k$ of resource k consumed when traversing the arc in the current relaxation, i.e., g_e^k if we track resource k on arc e and zero otherwise. Observe that we only track resource k on the arcs in the path (n_0, n_2, n_5, n_6). Table 2.1 lists all possible extensions of path P_0 to nodes in N_j^k with their cost and the amount of resource consumed on the path. Here, we denote with $g^k(P)$ and $\tilde{g}^k(P)$, the resource consumption along path P when using the actual values for resource consumption and the relaxed values for resource consumption, respectively. For simplicity, we assume that $c(P_0) = g^k(P_0) = 0$ and that the supports for any other resources do not contain nodes in N_j^k.

Assume that the resource limit is 40. When we examine the values for resource consumption for the paths in Table 2.1, we see that for the relaxed resource consumption there are 5 fewer state values leading to 5 fewer non-dominated paths. Moreover, the lowest cost extension, i.e., $P_{20} = (P_0, n_2, n_5, n_6)$, remains infeasible. On the other hand, any reduction in the size of the state space by scaling the resource consumption values uniformly over all arcs will lead to a relaxation in which P_{20} becomes feasible. Finally, note that the infeasible extension $P_{11} = (P_0, n_2, n_3, n_4)$ has become feasible for the given relaxation. However, since its cost is worse than the best feasible extension, the subpath (n_0, n_2, n_3, n_4) is not deemed critical for resource k.

In the example, there is only a modest reduction in the size of the state space. The reduction of the size of the state space comes primarily from paths ending at nodes n_3 and n_4. Observe that these two nodes are somewhat isolated from the arcs on the critical path (n_0, n_2, n_5, n_6). Indeed, only one of the arcs on the critical path, namely arc (n_0, n_2), can be part of an extension of P_0 to either n_3 or n_4. This suggests that a huge reduction in the size of the state space may be obtained when only a small

Table 2.1: The cost and amount of resource consumed for all extensions of path P_0 given in Figure 2.1 when using actual values for resource consumption and certain relaxed values.

path P	$c(P)$	$g^k(P)$	$\tilde{g}^k(P)$
$P_1 = (P_0, n_1)$	0.0	5	0
$P_2 = (P_0, n_2)$	-1.5	11	11
$P_3 = (P_0, n_1, n_2)$	-1.1	9	0
$P_4 = (P_0, n_1, n_3)$	-1.1	16	0
$P_5 = (P_0, n_2, n_3)$	-2.6	25	11
$P_6 = (P_0, n_1, n_2, n_3)$	-2.2	23	0
$P_7 = (P_0, n_1, n_4)$	-1.0	15	0
$P_8 = (P_0, n_2, n_4)$	-2.5	22	11
$P_9 = (P_0, n_1, n_2, n_4)$	-2.1	20	0
$P_{10} = (P_0, n_1, n_3, n_4)$	-2.6	33	0
$P_{11} = (P_0, n_2, n_3, n_4)$	-4.1	42	11
$P_{12} = (P_0, n_1, n_2, n_3, n_4)$	-3.7	40	0
$P_{13} = (P_0, n_1, n_5)$	-1.0	15	0
$P_{14} = (P_0, n_2, n_5)$	-2.5	26	26
$P_{15} = (P_0, n_1, n_2, n_5)$	-2.1	24	15
$P_{16} = (P_0, n_1, n_6)$	-3.0	16	0
$P_{17} = (P_0, n_2, n_6)$	-4.5	22	11
$P_{18} = (P_0, n_1, n_2, n_6)$	-4.1	20	0
$P_{19} = (P_0, n_1, n_5, n_6)$	-5.0	31	16
$P_{20} = (P_0, n_2, n_5, n_6)$	-6.5	42	42
$P_{21} = (P_0, n_1, n_2, n_5, n_6)$	-6.1	40	31
no. of feasible states (when $b^k = 40$)		19	14
no. of non-dominated feasible paths		19	14

proportion of nodes in the support of a given resource are reachable from arcs that are "critical" to ensuring feasibility of the optimal path with respect to the given resource. This is a property typically encountered in time-expanded networks.

Although, the example demonstrates the potential benefit from using an arc-based relaxation of resource consumption, it also raises the question of how to implement such a relaxation. Clearly, it is too ambitious to try to find a relaxation that leads to the smallest state space and that still guarantees feasibility of the optimal path found. A more pragmatic approach is to find a relaxation that guarantees feasibility of the optimal path found, but that leads to a more manageable state space. Intuitively, it seems possible to achieve this by only tracking the consumption of a given resource on arcs that are necessary to ensure that any infeasible path with cost better than an optimal feasible solution remains infeasible. We next lay the groundwork that establishes this relaxation.

For each arc $e \in A_j$ and $k = 1, \ldots, K$, let

$$\mathcal{F}(g_e^k) = \begin{cases} g_e^k & \text{if we track resource } k \text{ on arc } e, \text{ and} \\ 0 & \text{if we do not track resource } k \text{ on arc } e. \end{cases}$$

We use the terms $\mathcal{F}$-*feasible* and $\mathcal{F}$-*dominance* when referring to path feasibility and dominance when resource consumption is given by $\mathcal{F}(g_e^k)$ for each arc $e \in A$ and resource $k = 1, \ldots, K$, the term $\mathcal{F}$-*optimal* when referring to optimality with respect to $\mathcal{F}$-feasible paths, and we use $P_1 \preceq_{\mathcal{F}} P_2$ to denote $\mathcal{F}$-dominance of path P_1 over P_2. Note that for each $k = 1, \ldots, K$, the support N_j^k constructed using actual values for resource consumption remains a support for resource k in network $\mathcal{N}_j$ when measuring consumption with respect to $\mathcal{F}(g_e^k)$ instead of g_e^k. Indeed, any feasible path remains feasible when relaxing resource consumption. Thus, we can use the original supports to prove $\mathcal{F}$-dominance of one path over another.

In the remaining part of this section, we introduce algorithms and schemes that build on the ideas introduced in this subsection to direct a relaxation towards finding

the optimal feasible path while continuing to explore a smaller more manageable number of paths by not only maintaining a relatively small state space, but by also using the bounds produced as a natural byproduct of the relaxation to further prune the search.

2.4.3 A Forward and Backward Dynamic Program for $\mathcal{F}$-feasible RCSPP

Given a relaxation for RCSPP defined by $\mathcal{F}$ for all arcs $e \in A_j$ and $k = 1, \ldots, K$, a DP algorithm can be used to construct an $\mathcal{F}$-optimal path from source to sink. This path may or may not be feasible. If it is not, the relaxation can be tightened, by refining the discretization of resource consumption along certain arcs, to ensure that the same path will not be found when the DP algorithm is run again. By running the DP algorithm in the opposite direction, i.e., constructing an $\mathcal{F}$-optimal path from sink to source, and using information gathered in the forward pass, much more effective pruning can be performed. We describe the iterative process of progressively tightening the relaxation in detail in the next subsection, but first outline the workings of the DP for a single pass and show how one can obtain bounds as a natural byproduct of the relaxation and use them to significantly reduce the number of states explored.

Suppose that we have a relaxation of RCSPP defined by $\mathcal{F}$ for each $e \in A_j$ and each $k = 1, \ldots, K$, that we have lower-bounds $T(n)$ on the cost of a feasible path from node n to the sink, and that we have an upper-bound UB on the cost of a minimum-cost path from source to sink. Algorithm 1 outlines the Forward DP procedure `FwdDP` that constructs an $\mathcal{F}$-optimal path from source to sink. Starting with the source node n_j^s, the unprocessed list L containing the trivial path rooted at the source node, denoted by (n_j^s), and an empty processed list U, `FwdDP` picks a previously constructed path P from L, moves it to U, and extends it by each out-going arc $e' \in (\overrightarrow{n}(P), n') \in A_j$, where $\overrightarrow{n}(P)$ denotes the last node in P. If the newly constructed path (P, e') is either $\mathcal{F}$-infeasible, proved not to be feasibly extendable to the sink with cost less than UB,

or $\mathcal{F}$-dominated by some previously constructed path in L or U, it is immediately discarded. Otherwise, we remove from L and U any path that is $\mathcal{F}$-dominated by (P, e') and insert (P, e') into L for future extension. For any newly constructed path inserted into L, we update UB if the path also corresponds to a feasible (not just $\mathcal{F}$-feasible) path.

Input : $\mathcal{F}$, $\mathsf{T}(\mathsf{n})$ $\forall \mathsf{n} \in \mathsf{N_j}$ and UB
Initialize: $\mathsf{L} \leftarrow \{(\mathsf{n_j^s})\}$ and $\mathsf{U} \leftarrow \{\emptyset\}$;

```
1.1  while L ≠ {∅} do
1.2      pick P ∈ L;
1.3      forall e' = (n→(P), n') ∈ A_j do
1.4          if (P, e') is F−feasible then
1.5              if c((P, e')) + T(n') < UB then
1.6                  if P' ⋠_F (P, e') for any P' ∈ L ∪ U s.t. n→(P') = n' then
1.7                      remove from L and U any path P' s.t. n→(P') = n' and
                         (P, e') ⪯_F P';
1.8                      L ← L ∪ {(P, e')};
1.9                      if (P, e') is a feasible (not just F-feasible) path from
                         source to sink then
1.10                         UB ← c((P, e'));
1.11                     end
1.12                 end
1.13             end
1.14         end
1.15     end
1.16     U ← U ∪ {P};
1.17 end
```

Output : UB, $\mathsf{S}(\mathsf{n}) = \min\{\mathsf{c}(\mathsf{P}) : \mathsf{P} \in \mathsf{U} \text{ and } \overrightarrow{\mathsf{n}}(\mathsf{P}) = \mathsf{n}\}$ for all $\mathsf{n} \in \mathsf{N_j}$ and $\mathsf{P}^* = \arg\min\{\mathsf{c}(\mathsf{P}) : \mathsf{P} \in \mathsf{U} \text{ and } \overrightarrow{\mathsf{n}}(\mathsf{P}) = \mathsf{n_j^t}\}$

Algorithm 1: `FwdDP`: A Forward DP for $\mathcal{F}$-feasible RCSPP on $\mathcal{N}_j = (N_j, A_j)$.

Algorithm 1 is a standard DP procedure for RCSPP similar to the one outlined in Irnich and Desaulniers [77] with the exception that we measure feasibility and dominance with respect to the relaxation given by $\mathcal{F}$, and we prune the search using the bounds given by $T(n)$ for each node $n \in N_j$ and UB.

For any feasible path P_2 from the source to node n such that $c(P_2) + T(n)$ is

less than the cost of any feasible path that is already added to RMP, Algorithm 1 maintains a processed list of paths U and an unprocessed list of paths L such that at any point during the course of the DP, there always exists some $\mathcal{F}$-feasible path say P_1 from source to node n such that $P_1 \preceq_{\mathcal{F}} P_2$ and either $P_1 \in U$, or $P_1' \in L$ and P_1 can be obtained by an extension of P_1' (see related discussion in Irnich and Desaulniers [77]). Thus, assuming the algorithm indeed terminates, the output $S(n)$ for each node $n \in N_j$ is a lower bound on the cost of a feasible path from source to n.

As the name suggests, `FwdDP` constructs paths going forward from the source node. We can similarly state a symmetrical algorithm that starts from the sink node and works its way backwards pruning the search using the bounds $S(n)$ instead of $T(n)$ for each $n \in N_j$. `BwdDP` accepts as an input lower bounds $S(n)$ on the cost of a min-cost path from the source to node n for each node $n \in N_j$ and an upper bound UB on the minimum-cost path from source to sink. `BwdDP` outputs the $\mathcal{F}$-optimal path P^* constructed from sink to source, and bounds $T(n)$ for each node $n \in N_j$ for pruning the search in the opposite direction.

2.4.4 An Iterative DP-based Search Procedure for RCSPP

Algorithm 2 describes a search procedure `DPSearch` that combines the relaxation described in Section 2.4.2 within an iterative DP-based search procedure with alternating search direction (i.e., alternating between `FwdDP` and `BwdDP`). In each pass of the search, we use the bounds obtained from the previous pass to prune the search in subsequent passes. Furthermore, we can ensure that the $\mathcal{F}$-optimal path P^* from source to sink, if infeasible, does not reappear in subsequent passes. Suppose P^*, the path produced by either `FwdDP` or `BwdDP` is infeasible because $g^k(P^*) > b^k$ for some $k \in \{1, \ldots, K\}$. This can only happen if we do not track resource k on some of the arcs $e \in A(P^*)$. By tracking resource k for the appropriate arcs in P^*, we can ensure that the same infeasibility does not reappear.

Initialize: $\mathsf{UB} \leftarrow \infty$ and $\mathsf{LB} \leftarrow -\infty$;
$\mathsf{S}(\mathsf{n}) \leftarrow -\infty$ and $\mathsf{T}(\mathsf{n}) \leftarrow -\infty$ for all $\mathsf{n} \in \mathsf{N_j} \setminus \{\mathsf{n_j^s}, \mathsf{n_j^t}\}$;
$\mathsf{S}(\mathsf{n_j^s}) \leftarrow 0$ and $\mathsf{T}(\mathsf{n_j^t}) \leftarrow 0$;
$\mathcal{F}(g_e^k) \leftarrow 0$ for all $\mathsf{e} \in \mathsf{A_j}$ and $\mathsf{k} \in \{1, \ldots, \mathsf{K}\}$
$\mathsf{pass} \leftarrow 0$;

2.1 **while** $\mathsf{UB} - \mathsf{LB} > \varepsilon$ **do**
2.2 **if** pass *is even* **then**
2.3 $(\mathsf{UB}, \mathsf{S}, \mathsf{P}^*) = \texttt{FwdDP}(\mathcal{F}_{\mathsf{e}\in\mathsf{A_j}}, \mathsf{T}, \mathsf{UB})$;
2.4 **else**
2.5 $(\mathsf{UB}, \mathsf{T}, \mathsf{P}^*) = \texttt{BwdDP}(\mathcal{F}_{\mathsf{e}\in\mathsf{A}_j}, \mathsf{S}, \mathsf{UB})$;
2.6 **end**
2.7 $\mathsf{LB} \leftarrow \mathsf{c}(\mathsf{P}^*)$;
2.8 **if** P^* *is infeasible* **then**
2.9 $\mathcal{F}(g_e^k) \leftarrow g_e^k$ for one or more arcs $e \in A_j$ and one or more resources $k \in \{1, \ldots, K\}$ so that P^* is no longer $\mathcal{F}$-feasible;
2.10 **end**
2.11 $\mathsf{pass} \leftarrow \mathsf{pass} + 1$;
2.12 **end**

Output : P^*

Algorithm 2: `DPSearch`: A DP for RCSPP on $\mathcal{N}_j = (N_j, A_j)$ using `FwdDP`, `BwdDP` and relaxation.

The performance of Algorithm 2, in terms of speed and memory usage, depends greatly on the initial relaxation and the degree of tightening after each pass. If we start with a complete state space relaxation (i.e., no tracking of any resource), and in subsequent passes track only one of the resources that caused infeasibility of P^* and only on the relevant arcs in $A(P^*)$, then we increase the chance of a node in the support remaining isolated (reachable) from the arcs on which resources are being tracked leading to a small state space. However, such a strategy may require a large number of passes. Furthermore, with such a strategy, the bounds available for pruning the search in consecutive passes will only improve slightly. On the other hand, if we decide to track all the resources that caused infeasibility, also on arcs other than the ones in $A(P^*)$, and if in addition, we track resources other than those that caused infeasibility of P^*, then we are likely to need far fewer passes at the possible expense of exploring a larger state space in each of them, but with stronger bounds for pruning

the search in each of them as well. The resources to track in subsequent passes and the arcs on which to track them have to be chosen so as to (1) ensure that the infeasible $\mathcal{F}$-optimal path of the current pass does not reappear, and (2) balance the number of passes and the effort expanded per pass. A judicious choice may result in faster solution times and less memory use. In our computational study, we experiment with different choices to decipher the relative merits of tightening the relaxation over many resources and arcs versus tightening the relaxation more selectively over a few resources and a small set of arcs.

Besides solving a relaxed problem, it is important to note that the search procedure outlined in Algorithm 2 is not a bi-directional search in the traditional sense as described for example in Righini and Salani [92]. Rather than having two concurrent searches in which we can only fathom paths reaching some predefined "half-way" point, we alternate between forward and backward passes, allowing us to compute bounds to start pruning at any stage in the search. We also point out that our procedure is not simply an extension of the scaling algorithm of Dumitrescu and Boland [50] to multiple resources. In addition to obtaining bounds and pruning the search by alternating search directions, we are able to vary the size of the state space over the network by allowing refinement of the relaxation to specific resources and arcs. Furthermore, our bounds for pruning are obtained and strengthened as a natural byproduct of the relaxation scheme and refinement process without having to solve auxiliary optimization problems.

2.4.5 A Path Completion Heuristic

A drawback of the scheme just described is that we do not obtain the optimal feasible path until close to termination and in general, only obtain a handful of other feasible paths in the process. With this in mind, we next describe a simple heuristic that can be used within the DP algorithm described in Section 2.4.3. The benefit of this

heuristic is twofold: Not only does it give us a simple yet efficient means of populating RMP with more than a single column per pricing iteration, it also serves to improve the pruning within `FwdDP` and `BwdDP` due to improved upper-bounds UB obtained sooner.

In `FwdDP`, when inserting a newly constructed $\mathcal{F}$-feasible-non-dominated path P into the unprocessed list, we backtrack over P to check if it is feasible (not just $\mathcal{F}$-feasible) and if so, try and extend P to the sink via a Greedy Depth First Search (GDFS). The greediness of the algorithm stems from our choice of the arc to explore next. Consider an iteration of GDFS in which we are looking to extend path P' obtained from extending P. Let $\{e_1 = (\overrightarrow{n}(P'), n_1), e_2 = (\overrightarrow{n}(P'), n_2), \ldots, e_q = (\overrightarrow{n}(P'), n_q)\}$ be the subset of out-going arcs for which an extension of P' has not yet been explored in GDFS. Of these possible choices to extend P', we pick arc e_i such that $c_{e_i} + T(n_i) \leq c_{e_j} + T(n_j)$ for all $j = 1, \ldots, q$. Extending the search by outgoing arc e_i is more likely (from the point of view of the bounds) to result in a path that improves UB than any other arc e_j for $j \in \{1, \ldots, q\} \setminus \{i\}$. If the extension (P', e_i) is infeasible then we backtrack and iterate. Similarly, if $c((P', e_i)) + T(\overrightarrow{n}((P', e_i))) > UB$ then we also backtrack. However, in this case, we can backtrack twice before continuing the search since by our choice of arcs to explore next, we also have $c((P', e_j)) + T(\overrightarrow{n}((P', e_j))) > UB$ for all $j = i + 1, \ldots, q$.

Of course, it would be impractical to perform this search for every new path inserted into the unprocessed list and continue the search exhaustively until the best possible extension is found. Instead, we first check if path P has the potential to improve UB significantly, i.e., $c(P) + T(\overrightarrow{n}(P)) < LB - \theta(UB - LB)$ for some $0 < \theta < 1$, and stop after a certain number of extensions to the sink or after finding an extension that significantly improves UB. Of course, a symmetrical procedure can be used to extend newly constructed paths to the source node when in the backward pass.

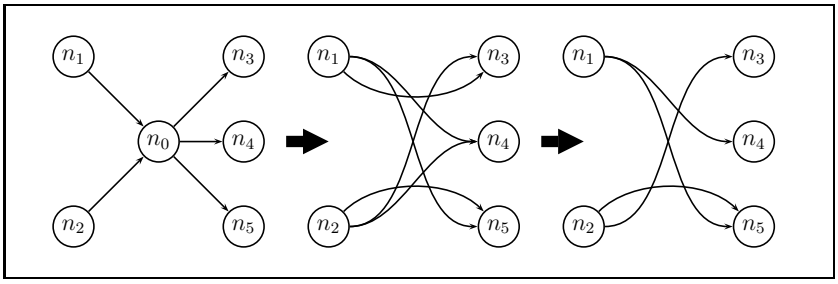


Figure 2.2: The aggregation algorithm: Node n_0 is aggregated; the resulting paths (n_1, n_0, n_3) and (n_2, n_0, n_4) are infeasible and discarded.

2.4.6 Network Preprocessing

We next describe the use of aggregation, a unique form of pre-processing that looks to remove certain infeasible paths from the network. Aggregation was originally introduced in Espinoza et al. [54] on a much smaller scale to strengthen a multi-commodity flow formulation. It is also related to adding infeasible path inequalities (see Ropke and Cordeau [94] and Spoorendonk and Petersen [105]) to the formulation a priori by modifying the network structure rather than actually adding cuts to the IP formulation. Although, aggregation does not strengthen the column generation formulation, it has a significant impact on the tractability of the pricing problem.

The process of aggregation is quite simple: we replace a node and its incoming and outgoing arcs with arcs from the tail of each incoming arc to the head of each outgoing arc as shown in Figure 2.2. When this process is applied iteratively, then the newly added arcs correspond to paths in the original network and thus, may be infeasible in terms of the resources accumulated and need not be added. Each time a node is aggregated, we remove a node from the network but may add quadratically more arcs than we remove. The key, therefore, is in the selection of the nodes to aggregate. Time-expanded networks usually contain substructures like forks, paths, and small knots that are ideal candidates for aggregation as shown in Figure 2.3. In our implementation of aggregation, we iterate until no such substructures can

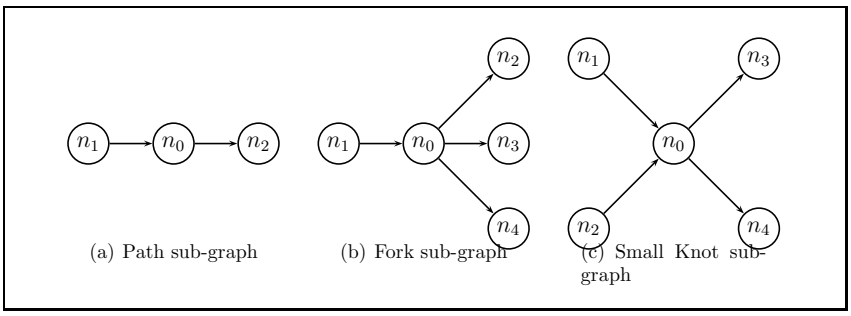


Figure 2.3: Network sub-structures ideal for aggregation.

be found. Note that these substructures are less likely to be present in the more traditional, compact representations used in the literature for VRPTW and PDPTW and thus aggregation may have little or no impact there.

2.5 Computational Experiments

In this section, we present computational experiments to evaluate the efficiency of the proposed scheme for solving RCSPP based column generation. We use "real-life" instances of DAFP provided by DayJet® Corp. (`www.dayjet.com`) and the network formulation proposed in Espinoza et al. [54] that exemplify the difficulty in solving RCSPP-based column generation in extremely large networks with many local resource constraints. We start by giving a brief overview of DAFP and highlight some characteristics of the network formulation that make RCSPP-based pricing particularly difficult. We then conduct a number of experiments to evaluate the impact of various algorithmic choices and conclude by presenting the actual bounds obtained by solving the column generation relaxation for these instances.

2.5.1 DAFP and RCSPP-based Column Generation

In DAFP, we have a set of jets and a set of requests for transportation from an origin airport to a destination airport. We have to build flight plans that ensure that each request is satisfied within a specified time window, that each passenger is picked up and dropped off within a certain amount of time, and that each passenger has at

most one intermediate stop. Each jet has a specific seating capacity and weight limit including the weight of the fuel it carries. Finally, the flying-time a crew may accrue is limited by enforcing a maximum flying-time over each shift.

In Espinoza et al. [54] the problem is modeled on an acyclic time-activity network in which all feasible passenger and jet itineraries can be represented by paths from the source to sink. Each node in this network corresponds to a point in time and space where a jet or a passenger associated with a particular request is located. Additionally, a node may correspond to a decision to relocate the jet to another location or to pickup passengers and transport them directly or indirectly to their final destination. Since the routing decisions for requests are explicitly considered within the network, an arbitrary path from source to sink satisfies the time window restrictions and the one intermediate stop requirement for each request, but it may violate the capacity and weight limit of the jet, it may exceed the maximum flying-time in a shift, and it may satisfy a request more than once. In the column generation formulation of the problem, we ensure that each request is picked up exactly once in the master problem and we solve RCSPP in the pricing problem to generate passenger and jet itineraries that are feasible for the capacity, weight, and flying-time restrictions, and that ensure each passenger is picked-up at most once.

The size of the networks and the number of resource constraints needed to ensure feasible itineraries in this formulation grow rapidly with number of requests and airports. This is despite the fact that the networks are meticulously constructed by including only nodes and arcs that can be part of some feasible itinerary and by eliminating redundancies, e.g., by including only a single representative of itineraries that differ only by departure times or by including only a single representative of itineraries that differ only in the sequence in which requests are picked-up for the same leg. This growth is to be expected for a time-activity network representation for a passenger transportation problem without an underlying schedule, since we need

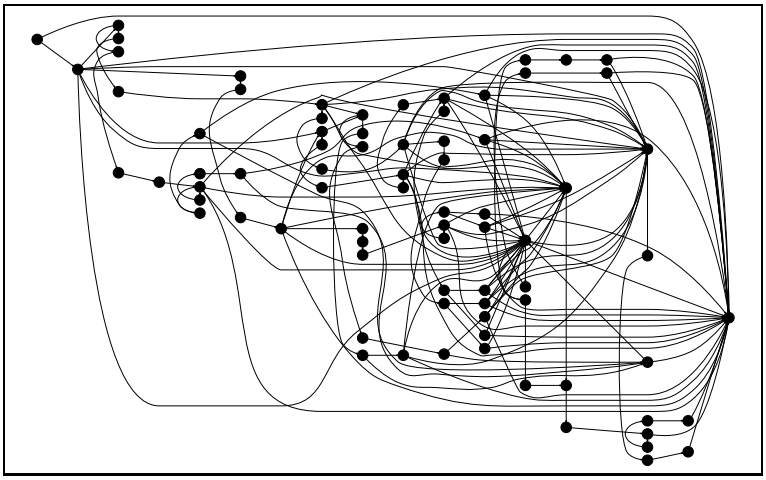


Figure 2.4: An example network constructed for an instance with 3 airports and 6 requests.

to consider all possible legs and departure times, and all possible itineraries (i.e., all routing possibilities) for each request. Figure 2.4 displays an example network for a single jet resulting from an instance with 3 airports and 6 requests. Even for this insignificantly small instance, the resulting network representation has 78 nodes, 167 arcs and 45 different constraints on resource consumption. Our challenge is to solve instances with 200 jets, 1600 requests, and 40 airports.

In all, we test our algorithms on a total of 50 instances ranging from 10 jets and approximately 60 requests up to 200 jets and approximately 1600 requests. Each instance is characterized by the number of jets, the number of requests, the number of airports, and the actual day for which the schedule is to be constructed. We denote, for example, by $(100J, 1200R, 40P, 10)$ an instance for day 10 with 100 jets, 1200 requests, and 40 airports. In each case, there is a homogeneous fleet of jets with seating capacity for three passengers and based at the same airport. All time windows and flying-times are stated in minutes and each instance represents the daily operation over two overlapping shifts spanning a 16-hour day. The objective in each case is to minimize the total flying-time (and thus fuel burn).

Table 2.2 gives the size of the network and various statistics relating to the number

of resource constraints and the size of supports for select instances. The first column lists the instance and the second column gives the number of nodes and arcs in the network. Since we assume a homogeneous fleet, we construct a single network and use it for the pricing of all jets. The remaining three columns, labeled "Capacity/Weight", "Flying-Time," and "Requests" correspond to the types of resource constraints within DAFP, i.e., enforcing the jet seating capacity on some leg, enforcing the jet weight limit on some leg, enforcing a maximum flying-time for some shift, and ensuring that some request is not picked-up more than once. For each instance and for each type of resource constraint, we display three pieces of information:

1. the number of constraints enforced,
2. the average size of the support, and
3. the average overlap of supports, i.e., number of supports that contain the same node.

The number of constraints required to enforce the jet capacity and the weight limit is exactly the number of legs within the network. Although there is a huge number of these constraints (over one million for the larger instances), the average size of the supports is extremely small (less than 10 nodes). Furthermore, the average overlap between these supports is never more than two. Indeed, passengers associated with a particular request can travel on at most two legs. Thus, any reasonably constructed support for the capacity or weight restriction on some leg should have no more than two such supports containing the same node when the node corresponds to picking-up passengers for some request. Note that as mentioned earlier, we do not construct minimal supports. This would be practically impossible when considering the size of the networks and number of resource constraints. For the capacity and weight constraints, we include a node in the support corresponding to a particular leg only if the node corresponds to a decision to relocate the jet or passengers on that leg. The

decisions to relocate a jet or passengers on the same leg appear contiguously over a path in the network. Thus, such inclusion is sufficient to construct the support for the capacity and weight constraints for each leg.

In contrast, we need only two constraints to enforce the restrictions on flying-time one for each of the two shifts. Each node in the network falls into exactly one shift, and nodes belonging to the same shift are contiguous over a path within the network. Thus, the average size of these supports is half the number of nodes in the network, and the average overlap between these supports is one.

Finally, we observe that the number of constraints required to enforce that a request is not picked up more than once is exactly the number of requests. Here a node is included as part of the support for not picking up a request more than once if the request can be part of an itinerary going through that node and still satisfy the time window of the request. The average overlap between such supports suggests that, on average, a request has the opportunity to be consolidated on the same leg with many other requests.

In summary, the instances and networks used to test our algorithms represent a diverse set of characteristics from network size to the types of resource constraints and size of the respective supports.

2.5.2 Computational Results

All our algorithms were implemented in C using CPLEX 9.1 barrier optimizer as the LP solver. Furthermore, all experiments were conducted on 2.4 GHz Dual AMD 250 processors with 4GB of RAM each.

2.5.2.1 Impact of aggregation on network characteristics

Figure 2.5 shows the network from Figure 2.4 after aggregation. Notice how aggregation dramatically reduces the number of nodes in this example and that the majority of resulting arcs are parallel and correspond to paths in the original network. In the

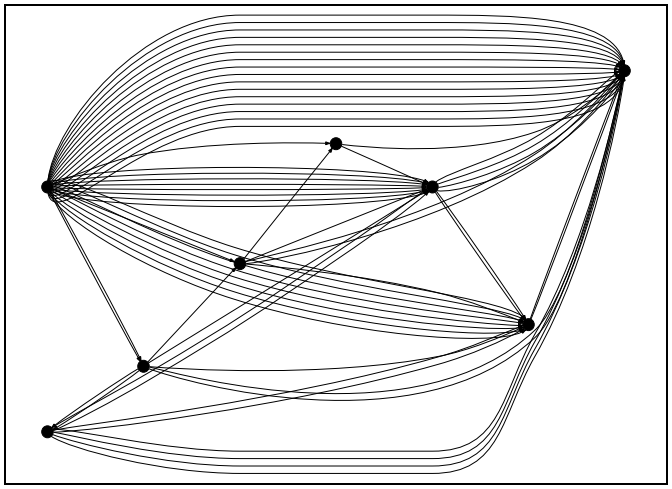


Figure 2.5: The result of aggregation on the network shown in Figure 2.4.

case of DAFP, the process of aggregation, when done intelligently, can be interpreted as grouping together sequences of "likely" decisions in the form of arcs within the aggregated network. The nodes that are not aggregated correspond to crucial decision epochs at which point the course of an itinerary can change dramatically.

Tables 2.2 and 2.2 shows for select instances the network characteristics before and after aggregation respectively. Observe that aggregation reduces the number of nodes in the network by a factor of at least 10, but that the number of arcs also decreases since many sequences of arcs can be eliminated based on resource considerations. At the same time, the average overlap between supports of the capacity and weight constraints increases by a factor of about 8 for the smaller instances and a factor of about 5 for the larger instances. The average overlap between the two supports corresponding to the flying-time restriction remains the same while the average overlap between supports corresponding to picking up a request no more than once increases by a factor of less than 2.

Through aggregation alone we can expect to see an order of magnitude decrease in the number of non-dominated paths stored at any given time. With a smaller network, we should also see a corresponding speedup in solution times. We quantify

this gain in efficiency through aggregation in the experiments that follow.

2.5.2.2 Impact of aggregation and the proposed relaxation scheme on the tractability of RCSPP

We attempt to solve the column generation relaxation for each of the 50 instances of DAFP provided by DayJet. Each instance is solved using networks with and without aggregation and using three different schemes, including two that use an arc-based relaxation of resource consumption.

The first scheme, the control experiment denoted by S^0, corresponds to a standard DP algorithm run with the proposed dominance scheme (see Section 2.4.1) but without any relaxation, thus guaranteeing the minimum-cost path to be found in a single pass.

The second scheme, denoted by S^1, corresponds to a strategy in which we start with a complete state space relaxation (i.e., we set the consumption of all resources on all arcs to 0) and we tighten the relaxation by selecting one resource k for which the $\mathcal{F}$-optimal path is infeasible, and track the consumption of resource k on all the relevant arcs in the infeasible $\mathcal{F}$-optimal path in all subsequent passes. In fact, if resource k corresponds to a capacity, a weight, or a flying-time constraint, then we strengthen the relaxation even further and track resource k including *all* other resources of the same type on *all* relevant arcs, not just those in the infeasible $\mathcal{F}$-optimal path, in all subsequent passes. The reason we do not choose to do the same for resource constraints corresponding to picking up a request at most once is that these constraints have a large overlap of supports, whereas the overlap between capacity and weight constraints is never more than two and between flying-time constraints never more than one. If there are several resources for which the $\mathcal{F}$-optimal path is infeasible, then we select resource k based on the frequency with which we encounter infeasibilities of this type throughout the column generation process. Thus, if capacity is the most frequently violated resource constraint on the $\mathcal{F}$-optimal paths encountered so

far, then we always give preference to refining the relaxation with respect to capacity first. Note that in S^1 we refine the relaxation over a large number of resources and arcs. Scheme S^1 thus, evaluates the merits of the proposed DP scheme when the mechanism for managing the number of paths being explored is biased towards finding stronger bounds for pruning the search over maintaining a small state space.

The third scheme, which we denote by S^2, corresponds to a strategy in which we proceed as in scheme S^1 until we detect that the number of paths being explored is growing too rapidly during some pass (and we may thus exhaust the available memory). In this case, we undo any refinement of the relaxation done at the end of the previous pass and re-start by tracking infeasibilities only on paths that provide the bounds used for pruning the search in the current pass. More precisely, given the set of processed paths U at the end of the previous pass, we identify the set of paths

$$\mathcal{P} = \left\{ P \in U : \begin{array}{l} g^k(P) > b^k \text{ for some } k, \text{ and} \\ c(P) = \left\{ \begin{array}{ll} S(\overrightarrow{n}(P)), & \text{if previous pass is } \texttt{FwdDP} \\ T(\overleftarrow{n}(P)), & \text{if previous pass is } \texttt{BwdDP} \end{array} \right. \end{array} \right\}.$$

That is, the set $\mathcal{P}$ is the set of all infeasible $\mathcal{F}$-optimal paths from either the source to all nodes or from all nodes to the sink. For each path $P \in \mathcal{P}$, we select one resource k for which P is infeasible, and track the consumption of resource k on all relevant arcs in $A(P)$ and in all subsequent passes. (Note that for a local resource constraint the number of "relevant arcs" in $A(P)$ may be quite small.) As before, if there are several resources for which P is infeasible, then we select resource k based on the frequency with which we encounter infeasibilities of that type. In any subsequent pass we continue to refine the relaxation by tracking the consumption of a resource on the relevant arcs of an infeasible minimum-cost path from either the source to some node or from some node to the sink.

Since our networks are acyclic, we do not need to explicitly store paths in the processed list since we explore them based on a topological ordering of nodes at which

they terminate. We keep a record of the number of unprocessed non-dominated paths stored at any given time within the DP and use this as an indicator of whether the number of states explored is growing too rapidly for the progress made in the DP and, if need be, switch strategies as prescribed in scheme S^2. Once the switch is made, it may require several further passes before the optimal feasible solution is found. To avoid the algorithm from stalling, we limit the total number of passes during each pricing iteration. Experiments using scheme S^2 with a limit of 10 passes produced good results. At the tenth pass, we simply revert to tracking all resources on all arcs and thus, are guaranteed to find the optimal feasible path. Of course, we still have the bounds from previous passes to prune the search and hence, although a significant proportion of pricing iterations (when using scheme S^2) actually reached this limit, we find that extending this limit served only to slow the overall pricing and did not allow us to solve larger instances. Thus, scheme S^2 is designed to be a compromise between fewer passes and providing stronger bounds for pruning the search at the possible expense of a larger state space, versus having a much smaller state space at the expense of providing weaker bounds for pruning the search and potentially requiring many more passes.

Figures 2.6 and 2.7 display the average time per pricing iteration (in seconds) and the percentage of available memory used during the DP algorithm as a function of the number of nodes in the original network representation for each of the 50 instances when solving the column generation relaxation using networks with and without aggregation and using the three schemes just described. Both figures demonstrate the value of the two main ideas, i.e., network aggregation and resource relaxation. Aggregation allows us to handle much larger networks without a noticeable decrease in the average time per pricing iteration (the largest instances solved with S^0 without aggregation are of size $2e^6$ and with aggregation are of size $4e^6$ and the average per iteration is about 35 seconds in both cases). Careful resource relaxation allows us to

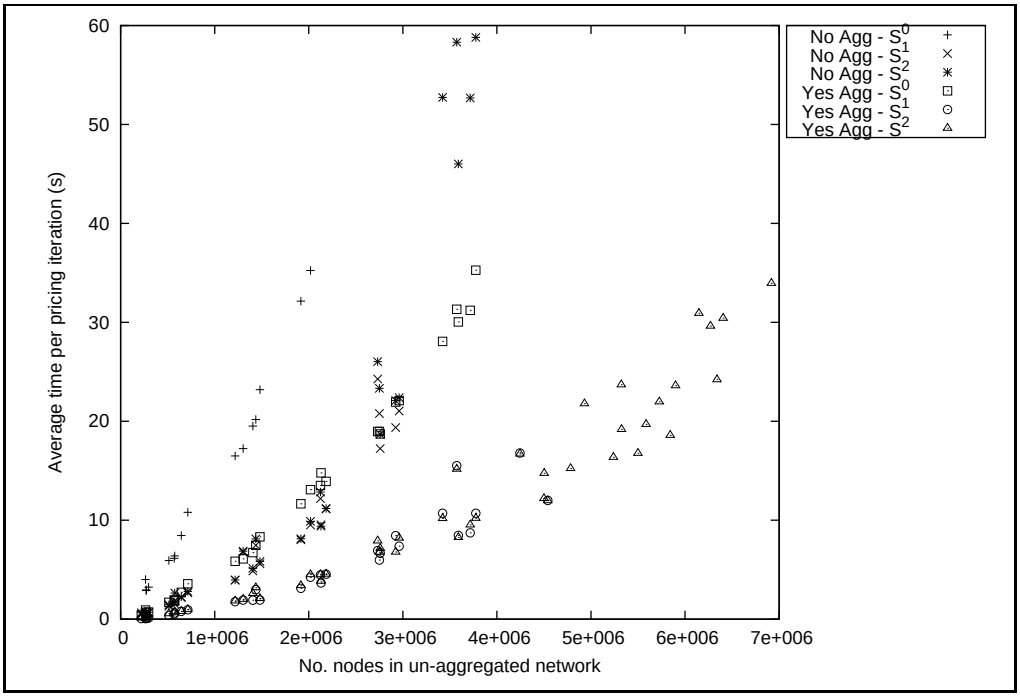


Figure 2.6: Average time per pricing iteration spent in pricing columns.

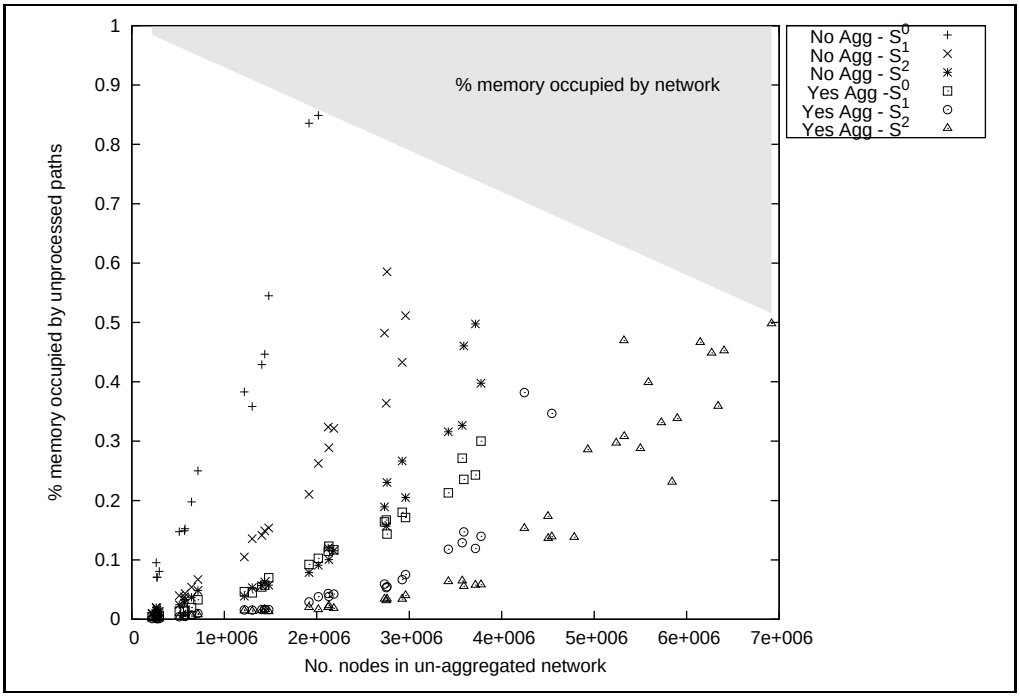


Figure 2.7: Percent of memory used by DP algorithm.

handle much larger networks without exhausting the available memory (with aggregation S^0 runs out of memory for networks of size $2e^6$, S^1 runs out of memory for networks of size $4.5e^6$, and S^2 runs out of memory for networks of size $7e^6$).

Tables 2.4 and 2.5 summarize the results over all 50 instances (10 different sizes and 5 instances per size) and the six experiments (the three schemes with and without aggregation) for each instance. The columns in Tables 2.4 and 2.5 are organized as follows. The first column gives the number of jets in the instance, the second column gives the number of airports, and the third column gives the number of requests (as a range). The next six columns summarize the results for six experiments carried out on each of these instances. The first three experiments correspond to the three schemes described when network aggregation is not performed whereas the last three experiments to the three schemes described when network aggregation is performed. Each cell of the last six columns presents six pieces of information:

1. the number of instances that could be solved,
2. the average of the time spent in pricing (in seconds),
3. the average number of pricing iterations,
4. the average number of passes per pricing iteration,
5. the average number of paths stored at any given time, and
6. the average of the maximum number of paths stored at any given time,

where the averages are taken over the solved instances only. Note that we do not impose any limits on solution time. Thus, experiments that failed to terminate exhausted the available memory, i.e., the memory required to store the network representation and the non-dominated paths became too large.

From Tables 2.4 and 2.5 we see that using schemes S^1 and S^2 are on average at least 3 times faster than the standard DP scheme S^0, and that using them in

conjunction with aggregation makes them 2 to 3 times faster again. The impact on the average and maximum number of paths stored at any given time follows a similar pattern. Although there is not much to separate schemes S^1 and S^2 in terms of pure speed, we notice that the average and worst-case number of paths stored at any given time is increasingly less for S^2 than for S^1. The true value of scheme S^2 is not evident until we examine the larger instances. The exponential growth of the state space becomes the main obstacle to solving these instances. It is clear that not only does it pay to use a relaxation scheme and prune the search using the bounds it provides, but that by judiciously selecting the arcs and resources we track once the state space hints at blowing up, we can solve much larger pricing problems.

In summary, standard DP techniques with the proposed dominance scheme allow us to consistently solve instances with up to 75 jets and 585 requests resulting in networks with approximately 2 million nodes, 4 million arcs, and over 1 million local resource constraints. With the addition of aggregation, we can increase this to instances with 125 jets and 966 requests resulting in networks with approximately 4 million nodes, 9 million arcs, and over 2 million local resource constraints. Finally, using an arc-based relaxation of resource consumption and a strategy that starts with the aim of producing strong bounds for pruning the search and later switches to a more conservative strategy in terms of choosing which arcs and resource to track, we are able to solve the column generation relaxation for instances with up to 200 jets and 1613 requests resulting in networks with approximately 7 million nodes, 16 million arcs, and over 4 million local resource constraints.

The schemes described here are only a few of the strategies we experimented with. Instead of starting with S^1 and switching to a more conservative strategy as prescribed in S^2, we could start with the conservative strategy right from the start. Of course with such a conservative scheme, one would expect many more passes and refinements per pricing iteration but fewer paths stored at any given time. However,

as with S^2, we need to impose a limit on the number of passes to stop the algorithm from stalling. We found that with this extremely conservative approach the algorithm almost always reached the limit on the number of passes even when extending the limit to several hundred per pricing iteration. Furthermore, since the bounds available for pruning are much weaker, the algorithm actually fared much worse than schemes S^1 and S^2 with respect to the number of instances that could be solved.

We also experimented with strategies in which we kept a history of arcs and resources that often contribute to infeasibility and use them to initialize the first pass or in addition to the arcs that contribute to the infeasibility in the current pass. We also considered the idea of initializing the tracking of resources on those arcs with favorable reduced cost. In each of these cases, we noticed a moderate speedup of the pricing process but could not solve instances larger than the ones already solved with the more simpler strategy S^2. Finally, we also experimented with progressively tightening the consumption of a resource along individual arcs rather than simply tracking a resource or ignoring it altogether. This had a modest impact on the size of the state space and the number of paths explored when applied to the weight and flying-time constraints. However, again we could not solve any larger instances.

2.5.2.3 Lower-bounds obtained from solving the column-generation relaxation

Tables 2.6 and 2.7 displays the bounds obtained for the 49 instances out of the 50 for which we were able to solve the column generation relaxation. Tables 2.6 and 2.7 give the instance, the value of the feasible schedule produced by DayJet's proprietary heuristic when run for 30 hours (an upper-bound), the value of the lower bound obtained by solving the column generation relaxation, and the gap between upper and lower bound. The gaps for the smaller instances (i.e., 10-50 jets and 60-360 requests) are comparable to what one may find in the literature for the gap for the bound found at the root node for related problems mentioned in Section 2.2. Of course

no comparisons are possible for the larger instances, but it is a pleasant surprise to see that the gaps do not grow as quickly as the size of the instances, and therefore remain meaningful for DayJet.

2.6 Conclusions

We have presented a relaxation-based dynamic programming algorithm for solving resource constrained shortest path problems arising in column generation pricing problems for formulations involving extremely large networks and a huge number of local resource constraints. The algorithm alternates between a forward and a backward search, and each search employs bounds derived in the previous search to prune the search. Between consecutive searches, the relaxation is tightened using a set of critical resources and a set of critical arcs over which these resources are consumed. As a result, a small state space can be maintained and many paths can be pruned while guaranteeing that an optimal path is ultimately found. Our computational experiments have demonstrated the efficacy of the proposed algorithm.

Table 2.2: Network characteristics for select instances before aggregation.

Instance	$\|N\|$	$\|A\|$	Type of resource constraint:		
			Capacity/Weight	Flying-Time	Requests
(10J,15P,60R,10)	223,860	374,370	120,972 2.54 1.37	2 111,930.00 1.00	60 33,389.35 8.95
(25J,20P,173R,10)	518,197	970,492	249,318 3.00 1.44	2 259,098.50 1.00	173 60,838.66 20.31
(50J,30P,356R,10)	1,238,059	2,499,362	527,273 3.55 1.51	2 619,029.50 1.00	356 157,285.83 45.23
(75J,30P,559R,10)	1,942,265	4,109,667	663,754 4.69 1.60	2 971,132.50 1.00	559 240,461.92 69.21
(100J,35P,751R,10)	2,788,465	6,215,328	844,351 5.44 1.65	2 1,394,232.50 1.00	751 350,087.97 94.29
(125J,41P,956R,10)	3,769,121	8,666,698	997,749 6.38 1.69	2 1,884,560.50 1.00	956 489,921.74 124.26
(150J,41P,1172R,10)	4,596,596	10,804,952	1,052,891 7.53 1.73	2 2,298,298.00 1.00	1,172 578,725.08 147.59
(175J,41P,1364R,10)	5,563,628	13,329,286	1,112,357 8.79 1.76	2 2,781,814.00 1.00	1,364 688,846.93 168.88
(185J,41P,1448R,10)	5,898,199	14,279,623	1,126,225 9.25 1.77	2 2,949,099.50 1.00	1,448 726,773.50 178.42
(200J,41P,1547R,10)	6,404,692	15,637,938	1,141,329 9.99 1.78	2 3,202,346.00 1.00	1,547 777,668.60 187.84

Table 2.3: Network characteristics for select instances after aggregation.

Instance	$\|N\|$	$\|A\|$	Type of resource constraint:		
			Capacity/Weight	Flying-Time	Requests
(10J,15P,60R,10)	8,925	110,907	71,951	2	60
			1.28	4,462.50	2,148.87
			10.32	1.00	14.45
(25J,20P,173R,10)	28,449	379,524	173,554	2	173
			1.52	14,224.50	5,178.71
			9.26	1.00	31.49
(50J,30P,356R,10)	85,044	1,142,065	418,624	2	356
			1.74	42,522.00	16,372.77
			8.55	1.00	68.54
(75J,30P,559R,10)	145,997	2,040,478	563,717	2	559
			2.28	72,998.50	27,718.89
			8.80	1.00	106.13
(100J,35P,751R,10)	236,137	3,330,594	756,251	2	751
			2.60	118,068.50	44,599.82
			8.32	1.00	141.84
(125J,41P,956R,10)	337,430	4,892,044	937,112	2	956
			2.99	168,715.00	66,357.31
			8.30	1.00	188.00
(150J,41P,1172R,10)	408,967	6,202,019	1,002,295	2	1,172
			3.47	204,483.50	77,774.37
			8.51	1.00	222.88
(175J,41P,1364R,10)	502,505	7,754,169	1,066,226	2	1,364
			4.04	251,252.50	92,926.22
			8.58	1.00	252.24
(185J,41P,1448R,10)	544,167	8,392,791	1,083,647	2	1,448
			4.25	272,083.50	100,683.83
			8.47	1.00	267.91
(200J,41P,1547R,10)	584,782	9,234,549	1,100,796	2	1,547
			4.58	292,391.00	106,944.50
			8.61	1.00	282.91

Table 2.4: A summary of the impact of aggregation and using an arc-based relaxation of resource consumption on solution time and problem tractability for instance with up to 100 jets.

No. Jets	No. Ports	No. Requests	No. Inst.	Aggregation: No			Aggregation: Yes		
				Scheme			Scheme		
				S^0	S^1	S^2	S^0	S^1	S^2
10	15	[60, 80]	5	5/5	5/5	5/5	5/5	5/5	5/5
				211	36	44	43	6	9
				69	69	70	65	74	67
				1.00	2.25	3.95	1.00	2.12	3.58
				760,088	145,349	133,879	60,944	8,175	7,167
				888,736	179,598	157,874	73,846	15,216	14,577
25	20	[173, 199]	5	5/5	5/5	5/5	5/5	5/5	5/5
				1,139	287	325	355	99	113
				147	146	147	145	148	147
				1.00	2.55	4.05	1.00	2.33	3.86
				1,681,698	336,950	258,609	178,164	31,180	21,566
				2,151,918	580,111	408,219	227,436	74,947	59,541
50	30	[350, 369]	5	5/5	5/5	5/5	5/5	5/5	5/5
				4,855	1,536	1,609	1,641	524	576
				250	269	271	238	248	245
				1.00	2.78	5.09	1.00	2.48	4.74
				3,979,851	730,913	485,845	462,409	73,908	61,847
				5,188,611	1,643,206	645,692	654,087	192,071	172,599
75	30	[543, 585]	5	2/5	5/5	5/5	5/5	5/5	5/5
				12,786	4,538	4,654	5232	1548	1612
				378	452	456	378	452	456
				1.00	2.88	5.23	1.00	2.75	4.97
				6,488,540	1,116,550	681,089	876,936	117,424	71,405
				10,106,232	3,377,045	1,212,930	1,319,072	459,434	234,420
100	35	[746, 779]	5	0/5	5/5	5/5	5/5	5/5	5/5
				-	14,714	16,034	9,872	3,549	3,568
				-	708	707	490	487	485
				-	2.87	5.52	1.00	2.51	5.23
				-	1,774,384	970,042	1,263,916	220,058	90,865
				-	5,700,995	2,515,464	1,983,181	742,298	416,281

Table 2.5: A summary of the impact of aggregation and using an arc-based relaxation of resource consumption on solution time and problem tractability for instance with up to 200 jets.

No. Jets	No. Ports	No. Requests	No. Inst.	Aggregation: No / Scheme			Aggregation: Yes / Scheme		
				S^0	S^1	S^2	S^0	S^1	S^2
125	41	[934, 966]	5	0/5	0/5	5/5	5/5	5/5	5/5
				-	-	33,882	18,618	6,474	6,315
				-	-	629	597	596	588
				-	-	6.12	1.00	2.57	5.52
				-	-	1,646,765	1,889,344	332,310	108,352
				-	-	4,794,583	3,031,568	1,567,290	718,551
150	41	[1135, 1182]	5	0/5	0/5	0/5	0/5	2/5	5/5
				-	-	-	-	10,511	9,935
				-	-	-	-	737	703
				-	-	-	-	2.68	5.64
				-	-	-	-	432,275	278,319
				-	-	-	-	4,370,936	1,776,275
175	41	[1312, 1382]	5	0/5	0/5	0/5	0/5	0/5	5/5
				-	-	-	-	-	15,778
				-	-	-	-	-	850
				-	-	-	-	-	5.97
				-	-	-	-	-	440,126
				-	-	-	-	-	3,382,715
185	41	[1385, 1487]	5	0/5	0/5	0/5	0/5	0/5	5/5
				-	-	-	-	-	20,456
				-	-	-	-	-	899
				-	-	-	-	-	6.31
				-	-	-	-	-	519,105
				-	-	-	-	-	4,552,260
200	41	[1516, 1613]	5	0/5	0/5	0/5	0/5	0/5	4/5
				-	-	-	-	-	31,382
				-	-	-	-	-	1,004
				-	-	-	-	-	6.97
				-	-	-	-	-	658,569
				-	-	-	-	-	5,596,450

Table 2.6: Lower bounds obtained for various instances with up to 100 jets.

Instance	$\overline{z}$	$\underline{z}$	gap (%)
(10J,60R,15P,10)	5,486	5,420	1.20
(10J,80R,15P,11)	6,941	6,784	2.26
(10J,72R,15P,12)	6,338	6,295	0.68
(10J,78R,15P,13)	6,959	6,959	0.00
(10J,71R,15P,14)	6,157	6,052	1.71
(25J,173R,20P,10)	17,449	17,355	0.54
(25J,199R,20P,11)	18,119	17,559	3.09
(25J,195R,20P,12)	18,253	17,628	3.42
(25J,176R,20P,13)	18,534	18,389	0.78
(25J,177R,20P,14)	19,078	18,708	1.94
(50J,356R,30P,10)	38,893	37,361	3.94
(50J,350R,30P,11)	38,074	36,813	3.31
(50J,359R,30P,12)	38,233	36,767	3.83
(50J,369R,30P,13)	38,123	37,150	2.55
(50J,360R,30P,14)	36,888	35,827	2.88
(75J,559R,30P,10)	57,758	55,350	4.17
(75J,552R,30P,11)	57,245	54,497	4.80
(75J,543R,30P,12)	55,225	52,430	5.06
(75J,585R,30P,13)	57,493	54,541	5.13
(75J,575R,30P,14)	55,423	53,858	2.82
(100J,751R,35P,10)	77,610	73,693	5.05
(100J,746R,35P,11)	76,402	73,697	3.54
(100J,750R,35P,12)	75,553	72,704	3.77
(100J,767R,35P,13)	73,624	70,286	4.53
(100J,779R,35P,14)	76,340	73,569	3.63

Table 2.7: Lower bounds obtained for various instances with up to 200 jets.

Instance	$\overline{z}$	$\underline{z}$	gap (%)
(125J,956R,41P,10)	95,681	91,925	3.93
(125J,966R,41P,11)	94,879	91,019	4.07
(125J,957R,41P,12)	92,903	88,402	4.84
(125J,934R,41P,13)	94,288	89,993	4.56
(125J,956R,41P,14)	95,487	91,669	4.00
(150J,1172R,41P,10)	113,931	108,481	4.78
(150J,1181R,41P,11)	112,908	107,256	5.01
(150J,1182R,41P,12)	110,924	105,266	5.10
(150J,1135R,41P,13)	112,190	107,204	4.44
(150J,1169R,41P,14)	114,130	109,228	4.30
(175J,1364R,41P,10)	131,770	124,424	5.57
(175J,1365R,41P,11)	129,121	121,214	6.12
(175J,1365R,41P,12)	130,095	121,162	6.87
(175J,1312R,41P,13)	129,323	121,733	5.87
(175J,1382R,41P,14)	131,900	124,982	5.24
(185J,1448R,41P,10)	138,122	130,075	5.83
(185J,1437R,41P,11)	136,091	127,863	6.05
(185J,1489R,41P,12)	137,391	127,880	6.92
(185J,1385R,41P,13)	134,376	126,221	6.07
(185J,1461R,41P,14)	137,143	129,507	5.57
(200J,1547R,41P,10)	149,036	138,913	6.79
(200J,1570R,41P,11)	147,369	136,403	7.44
(200J,1613R,41P,12)	148,585	137,974	7.14
(200J,1585R,41P,14)	148,192	138,868	6.29

CHAPTER III

THE FIXED CHARGE SHORTEST PATH PROBLEM

3.1 Introduction

Consider a network $\mathcal{N} = (N, A)$ where N is the set of nodes and A is the set of arcs. With each arc $e \in A$, we associate a fixed cost c_e for using arc e, an interval $[l_e, u_e]$ ($l_e, u_e \in \mathbb{Z}$) specifying the range of allowable resource consumption quantities along arc e, and a per unit cost $\bar{c}_e$ for resource consumed along e. Furthermore, for each node $n \in N$ we define $U_n \in \mathbb{Z}$ to be the maximum amount of resource consumption a path can accumulate before visiting node n. For a given path P, we use $A(P)$ to define the sequence of arcs traversed by P, $N(P)$ to be the sequence of nodes visited by P, and $W(P) = \{w_e : e \in A(P)\}$ to be the chosen consumption quantities along the arcs of P. The cost of a path P with consumption quantities $W(P)$ is given by $c(P, W(P)) = \sum_{e \in A(P)} (c_e + \bar{c}_e w_e)$ and the total amount of resource consumed along the path is given by $w(P, W(P)) = \sum_{e \in A(P)} w_e$.

Given a source node $n_s \in N$ and sink node $n_t \in N$, the *fixed charge shortest path problem* (FCSPP) can then be stated as

$$
\begin{aligned}
&\min\ c(P, W(P)) \\
&\text{s.t. } P \text{ is a path from } n_s \text{ to } n_t, \\
&\qquad l_e \le w_e \le u_e \text{ for all } e \in A(P), \text{ and} \\
&\qquad w(P', W(P')) \le U_n \text{ for all subpaths } P' \text{ of } P \text{ from } n_s \text{ to } n \in N(P).
\end{aligned}
$$

Note that the *resource constrained shortest path problem* (RCSPP) is the special case of FCSPP in which $l_e = u_e$ for all $e \in A$.

FCSPP appears naturally as part of the pricing process of column generation formulations for several important classes of scheduling, transportation, and resource allocation problems in which the pricing problem couples together discrete decisions with knapsack type constraints on continuous variables. Examples include the split delivery vehicle routing problem with time windows (SDVRPTW) (Dror et al. [47], Gendreau et al. [65], and Desaulniers [39]), the parallel machine scheduling problem with controllable processing times (PMSPCPT) (Nowicki and Zdrzalka [87], Cheng et al. [25], and Daniels and Sarin [36]), and inventory routing problems (IRP) (Federgruen and Zipkin [56], Campbell et al. [21], and Christiansen [26]).

Like traditional vehicle routing problems, SDVRPTW involves finding vehicle routes serving a set of customers from a depot. However, unlike traditional vehicle routing problems, we have flexibility in choosing the amount of a customer's demand that is satisfied on a particular route. In the natural column generation formulation for SDVRPTW, the master problem ensures that the total demand for each customer is satisfied while the resulting pricing problem involves finding a sequence of customers that are to be visited by a vehicle as well as the amount of the customer's demand that is satisfied on the given route. The typical network representation for vehicle routing problems includes nodes corresponding to customers (or the depot) and arcs corresponding to traveling from one customer to another. Here, vehicle capacity is a resource that is consumed along a path and in the case of SDVRPTW, we also have a choice in deciding how much of this resource is consumed by appropriately choosing the amount of a customer's demand that is to be satisfied when visiting the corresponding node. Thus, the bound limiting the total consumption of this resource up to a node corresponds to the capacity of the vehicle, and the amount of resource that is consumed when traversing an arc is bounded between 0 and the total demand associated with the customer corresponding to the head of the arc. Furthermore, in addition to fixed transportation costs on each arc, we have a per unit cost associated

with the dual value corresponding to the constraint in the master problem that ensures that all of a customers demand is ultimately satisfied (see [39]). Thus, the pricing problem involves finding a minimum reduced cost path that starts and ends at the node corresponding to the depot as well as the quantities of resource consumed when traversing the arcs of the path.

A well-known parallel machine scheduling problem involves scheduling a set of jobs, each with a given processing time and due date, on a set of identical machines so as to minimize the total tardiness. In the related PMSPCPT, the jobs must be completed by their due dates, but their processing time can be reduced at a cost. Thus, the goal is to find a schedule respecting the due dates at minimal cost. In the natural column generation formulation for PMSPCT, the master problem ensures that each job is performed while the pricing involves finding a sequence of jobs to be processed on a machine as well as the time taken to process each of these jobs in order to satisfy job due dates. The typical network representation for machine scheduling problems uses nodes corresponding to jobs and arcs corresponding to sequencing one job immediately after another. Here time is the resource that is consumed along a path and in the case of controllable processing times, we also have a choice in deciding the amount of this resource that is consumed by appropriately choosing the amount of time spent processing a job. Thus, the bounds limiting the total consumption of this resource up to a node correspond to the due date associated with the job corresponding to the node, and the amount of resource that is consumed when traversing an arc is bounded between 0 and the nominal processing time associated with the job corresponding to the head of the arc. Note that unlike SDVRPTW, the bounds on the consumption of the resource correspond to job due dates that may vary by job and thus by node visited.

Given a set of supply and demand points, the IRP seeks to minimize the cost of transporting inventory from supply to demand points so that supply/demand points

never exceed storage capacity or experience stock-outs. A possible column generation formulation for IRP which is also alluded to in Campbell et al. [21], includes a master problem in which inventory constraints for individual supply/demand points are used to ensure inventory is never depleted or that inventory never exceeds available storage capacity. Here the pricing problem involves finding vehicle routes between multiple supply and demand points as well as determining the timing and amount of inventory loaded/unloaded at each supply/demand point that is visited. In addition to ensuring vehicle capacity is not exceeded, we have to also ensure that no inventory is left on the vehicle at the end of its route. In a network representation where nodes correspond to supply/demand points and arcs correspond to the transition between such locations, the pricing problem involves finding a minimum reduced cost path in the network where in addition to fixed transportation costs, we have per unit costs associated with the dual values corresponding to the inventory constraints in the master problem for each supply/demand point. Vehicle capacity is the available resource and we have a choice in the amount of this resource that is consumed/replenished by appropriately choosing the amount of inventory that is loaded/unloaded at a supply/demand point when visiting the corresponding node. The upper bound on the total amount of resource consumed up to each node corresponds to vehicle capacity. However, since the resource is both consumed and replenished, we also need to impose a lower bound of zero to ensure that the amount of inventory unloaded at a demand point is never more than the amount of inventory on board the vehicle. Furthermore, to ensure no inventory is left on the vehicle at the end of its route, we also need to impose a lower and upper bound of zero at the sink node. Finally, the amount of resource consumed/replenished when traversing an arc is bounded between 0 and the capacity of the facility (or some bound on the maximum amount that can be loaded/unloaded during a visit to the facility) corresponding to the head node.

Column generation formulations give rise to some of the most successful exact

approaches for solving routing and scheduling problems, including the traditional variants of the problems discussed above. In these setting, the pricing is simply a RCSPP. Examples include, the vehicle routing and pickup and delivery problems without split deliveries or inventory management (see [34] and [33] for a survey including branch-price-and-cut approaches), parallel machine scheduling problems (Van den Akker et al. [108] and Chen and Powell [24]), the bin-packing problem (Valerio de Carvalho [107]), the cutting-stock problem (Vance et al. [110]), and the generalized assignment problem (Savelsbergh [98]). Since the variants discussed above represent a significant step towards reality as compared to their more traditional counterparts, it is somewhat surprising that the use of column generation with FCSPP has to date been limited to SDVRPTW, where the resulting pricing problem is a special case of FCSPP in which a single bound is imposed on the total amount of resource that can be consumed from source to sink (see Desaulniers [39]). For the few other examples of that do exist, the pricing problem is often transformed to resemble a more traditional RCSPP either through some artificial discretization of resource consumption at the expense of solution optimality (see for example Christiansen and Nygreen [29]), or by moving the decisions corresponding to resource consumption to the master problem resulting in a much weaker formulation (see for example Gendreau et al. [65]).

In this chapter, we develop a unique dynamic programming algorithm for FCSPP. The algorithm outlined in Section 3.3 uses for the most part, solutions from labeling and dominance techniques for standard RCSPP on slightly modified problems, and combines these solutions by exploiting structure provided by certain classes of knapsack problems described in Section 3.2 to efficiently construct an optimal solution to FCSPP. By exploiting the specific structure of the problem, our algorithm is in many cases better equipped to deal with the pseudo-polynomial nature of DP algorithms for these problems. This is reflected in our computational results in which our algorithm is often several orders of magnitude faster than more naive DP procedures.

We present computational results in Section 3.4 that demonstrate the effectiveness of the proposed algorithm on the pricing process for several classes of problems including SDVRPTW, PMSPCPT, and instances of a multi-period IRP. Together, these instances capture the complexities of the problems mentioned thus far and provide a basis to measure not only the viability of the proposed algorithm for use in column generation but also, its performance relative to the naive DP procedures.

3.2 Problem Characteristics

For a given path P, let $A(P) = \{e_1, e_2, \ldots, e_K\}$ and $N(P) = \{n_0, n_1, n_2, \ldots, n_K\}$. An optimal allocation of resource consumption $W^*(P) = \{w^*_{e_1}, w^*_{e_2}, \ldots, w^*_{e_K}\}$ can be obtained by solving the linear program:

$$\begin{aligned} \min \quad & \sum_{i=1,\ldots,K} \bar{c}_{e_i} w_{e_i} \\ \text{s.t.} \quad & \sum_{i=1,\ldots,j} w_{e_i} \leq U_{n_j} \text{ for all } j = 1, \ldots, K \text{ and} \\ & l_{e_i} \leq w_{e_i} \leq u_{e_i} \text{ for all } i = 1, \ldots, K. \end{aligned}$$

A closed form expression for $W^*(P)$ can be obtained by adapting the results of Faaland [55] and Dudzinski and Waluiewicz [49] for the LP relaxation of the *multi-period knapsack problem* for the case when the lower bound l_e may be non-zero and $\bar{c}_e$ may be positive.

Proposition 3.2.1. *Given a path P, $A(P)$, and $N(P)$, the optimal allocation (if one exists) of resource consumption $W^*(P) = \{w^*_{e_1}, w^*_{e_2}, \ldots, w^*_{e_K}\}$ is given by*

$$w^*_{e_k} = \min \left\{ \nu_{e_k}, \min_{i=k,\ldots,K} \left\{ U_{n_i} - \sum_{\{q \leq i : \bar{c}_{e_q} \leq \bar{c}_{e_k}\}} w^*_{e_q} - \sum_{\{q \leq i : \bar{c}_{e_q} > \bar{c}_{e_k}\}} l_{e_q} \right\} \right\}$$

for $k = 1, \ldots, K$ and

$$\nu_e = \begin{cases} l_e, & \text{if } \bar{c}_e \geq 0 \text{ and} \\ u_e, & \text{otherwise.} \end{cases}$$

Proposition 3.2.1 essentially states that the optimal allocation can be obtained by initializing consumption to the lower limit l_e for each arc $e \in A(P)$ and by increasing consumption on arcs greedily (i.e., in non-increasing order of $\bar{c}_e$) until we either reach the upper limit on resource consumption for the arc or we reach a bound on accumulated resource consumption. Clearly, once a bound on resource consumption accumulated up to a node is reached, no more resource can be consumed before this node. Thus, using this greedy approach, we can find an optimal allocation of resource consumption such that we can divide P into arc-disjoint sub-paths so that the start and end of each of these sub-paths is either the source, sink or some node at which the bound on resource consumption accumulated starting from the source up to that node is reached. Furthermore, each of these sub-paths has at most one arc with resource consumption strictly between its lower and upper limits. This leads us to the following observation that is integral to the design of our dynamic programming algorithm.

Corollary 3.2.2. *Given a path P, $A(P)$ and $N(P)$, if there exists a feasible allocation of resources along P then there exists an optimal allocation $W^*(P)$ such that for each $e_k \in A(P)$, either*

1. $w^*_{e_k} \in \{l_{e_k}, u_{e_k}\}$, *or*
2. $w^*_{e_k} = U_{n_i} - \sum_{j \in \{1,\ldots,i\} \setminus \{k\}} w^*_{e_j}$ *for some* $n_i \in N(P)$ *such that* $i \geq k$ *and* $w^*_{e_j} \in \{l_{e_j}, u_{e_j}\}$ *for all* $j \in \{k+1, \ldots, i\}$.

If a bound on accumulated resource consumption is enforced only for the sink node, i.e., $U_n = \infty$ for all $n \in N \setminus \{n_t\}$, Corollary 3.2.2 then states that there exists an optimal allocation of resources in which at most one of the arcs has resource consumption strictly between its lower and upper limits. Thus in the case of SD-VRPTW, we have at most one customer whose demand is fractionally satisfied (i.e.,

amount satisfied is strictly between 0 and customer's demand) on the route. This observation forms the basis of the DP algorithm developed by Desaulniers [39] for the pricing for SDVRPTW. He handles the decision to split a customer's demand as an additional resource in his labeling algorithm and the actual amount of split is not computed until a label is completed and corresponds to a complete route. As a result, in addition to dominance with respect to the split decision, his dominance scheme includes comparison of linear functions that measure the possible impact of a split decision on the future cost of a completed route. In the case when we have multiple non-uniform bounds on nodes and/or resource consumption along arcs can be both consumed and replenished, one does not know a priori the number of arcs in the optimal solution with resource consumption strictly between its lower and upper limit. Our DP approach decomposes the problem into several smaller RCSPPs and combines their solutions to produce an optimal solution to FCSPP by exploiting the structure provided by Corollary 3.2.2.

3.3 A Dynamic Program Labeling Algorithm for FCSPP

If $l_e = u_e$ for all $e \in A$, then w_e is fixed for all arcs and the problem is simply an RCSPP that can be solved by standard dynamic programming techniques (see for example Desrosiers et al. [45] and Irnich and Desaulniers [77]). In this case, a labeling algorithm is typically used in which a feasible path from the source to some node $n \in N$ is represented by a label $L = (n, c, w)$ where c is the cost associated with the path, w is the total amount of resource consumed along the path and a label $L_1 = (n_1, c_1, w_1)$ is said to dominate label $L_2 = (n_2, c_2, w_2)$ ($L_1 \preceq L_2$) if:

$$\text{(i) } n_1 = n_2, \text{ (ii) } c_1 \leq c_2, \text{ and (iii) } w_1 \leq w_2.$$

If $L_1 \preceq L_2$, then any feasible extension to the sink of a feasible path corresponding to label L_2 cannot lead to a better solution than the same extension of a feasible path corresponding to label L_1.

Analogous dominance can also be identified for the backwards recursion of the dynamic program, i.e., a dynamic program that starts from the sink and works it way backwards to the source node. In the backwards recursion, a feasible path from some node n to the sink is represented by a label $R = (n, c, s)$ where c is the cost associated with the path and s is the maximum amount of resource consumption that can be accumulated before visiting node n and still not exceed any bound on resource consumption accumulated along the path, i.e., given path P from n to the sink, $s = \min_{k=0,\ldots,K}\{U_{n_k} - \sum_{i=1,\ldots,k} w_{e_i}\}$. In this case, given two labels $R_1 = (n_1, c_1, s_1)$ and $R_2 = (n_2, c_2, s_2)$ for the backwards recursion, R_1 is said to dominate R_2 ($R_1 \preceq R_2$) if:

$$\text{(i) } n_1 = n_2, \text{ (ii) } c_1 \leq c_2, \text{ and (iii) } s_1 \geq s_2.$$

If $R_1 \preceq R_2$, then any feasible extension to the source of a feasible path corresponding to label R_2 cannot lead to a better solution than the same extension of a feasible path corresponding to label R_1.

Thus, starting with either the label corresponding to the trivial path rooted at the source node or trivial path rooted at the sink, a labeling algorithm at each iteration picks an unprocessed label, extends it by each outgoing arc (or each incoming arc in the case of the backwards recursion) and marks any new non-dominated labels as unprocessed for future extension. This way, the algorithm generates all non-dominated labels either working its way to the sink or working its way backwards to the source. The reader is referred to [77] for a more formal explanation of the labeling and dominance procedures for RCSPP.

Clearly, both forms of dominance are still valid when $l_e < u_e$ for one or more arcs $e \in A$. Although in this case, a label is determined not only by the path, but also by the chosen resource consumption along individual arcs of the path. As a result, unlike RCSPP, there are possibly uncountably many labels for FCSPP. However, using the property established in Corollary 3.2.2, we are able to construct an optimal solution

using dynamic programming over a pseudo-polynomial state space. If all data relating to resource consumption is integer, then from Corollary 3.2.2 we have that there exists an optimal solution in which resource consumption is also integer. Thus, by replacing each arc $e \in A$ within $\mathcal{N}$ with $u_e - l_e + 1$ arcs that consume exactly $l_e, l_e+1, l_e+2, \ldots, u_e$ units of resource respectively, we can thus reduce FCSPP to RCSPP naively and solve it using standard techniques. However, we can be more intelligent by exploiting more than simply the integrality property of resource consumption.

In the remaining part of this section, we show how FCSPP can be solved by solving several RCSPPs using a backwards recursion and then combining the non-dominated labels associated with these solutions within a single forward recursion. We start by demonstrating the procedure on a special case of FCSPP related to the pricing problem of SDVRPTW. This forms the building blocks for the DP of the more general case.

3.3.1 The Case of a Single Bound on Resource Consumption

Consider the case of FCSPP in which a single bound is imposed on the total amount of resource consumption accumulated along a path from source to sink, i.e., $U_n = \infty$ for all $n \in N \setminus \{n_t\}$. In this case, it follows from Corollary 3.2.2 that there exists an optimal solution $(P^*, W^*(P^*))$ in which $l_e < w_e^* < u_e$ for at most one of the arcs $e \in A(P^*)$. If $w_e^* \in \{l_e, u_e\}$ for all $e \in A(P^*)$, then the problem can be reduced to a standard RCSPP in a slightly modified network $\overrightarrow{\mathcal{N}} = (N, \overrightarrow{A})$ obtained by simply replacing each arc $e \in A$ by two arcs between the same pair of nodes; the first arc consuming exactly l_e units of resource and having a fixed cost $c_e + \bar{c}_e l_e$, and the second arc consuming exactly u_e units of resource and having a fixed cost $c_e + \bar{c}_e u_e$ (see Figure 3.1 (a) and (b)).

On the other hand, if there exists one arc $\bar{e} = (n_1, n_2) \in A(P^*)$ such that $l_{\bar{e}} < w_{\bar{e}}^* < u_{\bar{e}}$, then by removing $\bar{e}$ from P^*, we are left with two paths P_1^* from the source to

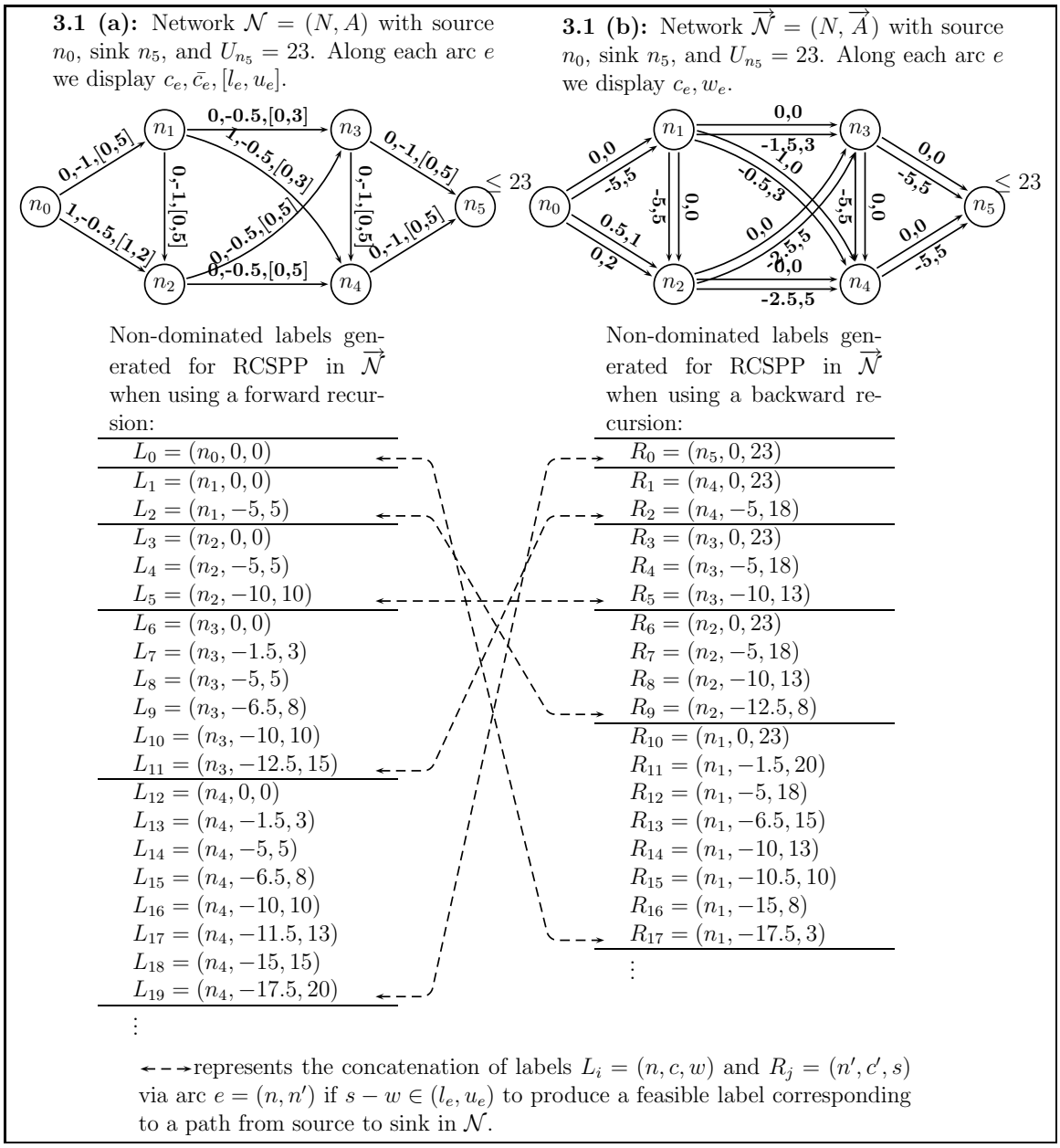


Figure 3.1: An example demonstrating the solution to FCSPP for the special case when a single bound is imposed on resource consumption.

n_1 and P_2^* from n_2 to the sink such that $w_e^* \in \{l_e, u_e\}$ for all $e \in A(P_1^*) \cup A(P_2^*)$. Since $(P^*, W^*(P^*))$ is optimal, $c(P_1^*, W^*(P_1^*))$ must be the minimum cost of a path from source to node n_1 in $\overrightarrow{\mathcal{N}}$ that does not consume more than $w(P_1^*, W^*(P_1^*))$ amount of resource. Similarly, $c(P_2^*, W^*(P_2^*))$ must be the minimum cost of a path in $\overrightarrow{\mathcal{N}}$ from n_2 to the sink that can feasibly accommodate at least an additional $U_{n_t} - w(P_2^*, W^*(P_2^*))$ units of resource consumption before visiting node n_2. The label associated with $(P^*, W^*(P^*))$ can then be obtained by concatenating the appropriate non-dominated labels generated by solving the standard RCSPP in network $\overrightarrow{\mathcal{N}}$ using the forward recursion with the non-dominated labels generated by solving RCSPP in $\overrightarrow{\mathcal{N}}$ using the backward recursion. Note that we only need to consider the concatenation of a label $L = (n, c, w)$ corresponding to a path from source to n and a label $R = (n', c', s)$ corresponding to a path from n' to sink if $l_e < s - w < u_e$ for $e = (n, n')$. Indeed, for $s - w < l_e$ the concatenation will lead to an infeasible path. For $s - w = l_e$ or $s - w \geq u_e$ the two labels can be feasibly concatenated however, the resulting label can also be obtained by extending L along e and any path corresponding to R and only considering consumption at either the lower or upper limit of each of these arcs, i.e., using standard RCSPP in $\overrightarrow{\mathcal{N}}$.

Consider the example of FCSPP given in Figure 3.1 where the problem is to find the min cost path from n_0 to n_5 that does not exceed 23 units of resource consumption. The optimal solution $(P^*, W^*(P^*))$ is the path $(n_0, n_1, n_2, n_3, n_4, n_5)$ with resource consumption $(5, 5, 3, 5, 5)$ on the respective arcs. The corresponding label associated with this solution is $L^* = (n_5, -21.5, 23)$. Note that resource consumption is at the upper limit for all the arcs except the arc (n_2, n_3) for which it is strictly between the lower and upper limit. Furthermore, the path consumes 10 units of resource before visiting node n_2 and 10 units of resource after visiting node n_3. Thus, L^* can be obtained by concatenating the min cost label in $\overrightarrow{\mathcal{N}}$ associated with n_2 that does not consume more than 10 units of resource with the min cost label in $\overrightarrow{\mathcal{N}}$ associated with

node n_3 that can accommodate at least an additional $23 - 10 = 13$ units of resource before entering n_3. These are exactly the labels L_5 and R_5 respectively. Thus, the problem can be solved by solving standard RCSPP in $\overrightarrow{\mathcal{N}}$ twice, once with the forward recursion and once with backward recursion and further examining all appropriate pairs of non-dominated labels that are separated by an arc for possible concatenation.

3.3.2 The General Case of Multiple Bounds on Resource Consumption

If in addition to the sink, we have other nodes in the network for which a bound exists on the amount of resource consumption a path can accumulate before visiting the node, then a simple concatenation of forward paths from the source and backward paths from the sink in $\overrightarrow{\mathcal{N}}$ may not be enough to ensure finding an optimal solution. Consider again the example given in Figure 3.1. If we reduce U_{n_5} to 21 from 23 and impose an additional bound $U_{n_3} = 12$ at n_3, then the optimal solution would be along the same path $(n_0, n_1, n_2, n_3, n_4, n_5)$ as before but with resource consumption $(5, 5, 2, 4+\theta, 5-\theta)$ on the respective arcs for $0 \leq \theta \leq 1$. For all $0 \leq \theta \leq 1$, at least two of the arcs have resource consumption strictly between their lower and upper limit and thus, an optimal solution cannot be obtained as before. Fortunately, Corollary 3.2.2 provides sufficient structure to exploit for this more general case.

Consider a path P from source to sink with $N(P) = \{n_0, n_1, \ldots, n_K\}$, $A(P) = \{e_1, e_2, \ldots, e_K\}$, and resource consumption $W(P)$ satisfying conditions of Corollary 3.2.2. We show that one can obtain a label corresponding to a path from source to sink that dominates the label corresponding to $(P, W(P))$ by only considering non-dominated labels from various recursions of standard RCSPP in $\overrightarrow{\mathcal{N}}$ and appropriate concatenations of these labels. Indeed, if $w_{e_j} \in \{l_{e_j}, u_{e_j}\}$ for all $j = 1, \ldots, K$, then a label corresponding to a path from source to sink that dominates the label corresponding to P can be obtained by solving standard RCSPP in $\overrightarrow{\mathcal{N}}$. Otherwise, let e_{k_1} be the first arc of P such that $l_{e_{k_1}} < w_{e_{k_1}} < u_{e_{k_1}}$. From Corollary 3.2.2 it

follows that there exists $i_1 \geq k_1$ such that $\sum_{j=1,\ldots,i_1} w_j = U_{n_{i_1}}$ and $w_j \in \{l_{e_j}, u_{e_j}\}$ for all $j = k_1 + 1, \ldots, i_1$. Furthermore, since e_{k_1} is also the first arc in P where resource consumption is strictly between its lower and upper limits, we must also have $w_j \in \{l_{e_j}, u_{e_j}\}$ for all $j = 1, \ldots, k_1 - 1$. Let w_L be the amount of resource consumed along P from n_0 to n_{k_1-1} and w_R be the amount of resource consumed along P from n_{k_1} to n_{i_1}. Hence, the label corresponding to the sub-path of P from n_0 to n_{i_1} is dominated by a label say L_2 obtained by concatenating the min cost label corresponding to a path from n_0 to n_{k_1-1} in $\overrightarrow{\mathcal{N}}$ that does not consume more than w_L units of resource with the min cost label from n_{k_1} to n_{i_1} in $\overrightarrow{\mathcal{N}}$ that can accommodate at least a further $U_{n_{i_1}} - w_R$ units of resource before visiting node n_{k_1}. Note that a path in $\overrightarrow{\mathcal{N}}$ from n_{k_1} to n_{i_1} that can accommodate at least $U_{n_{i_1}} - w_R$ units of resource before arriving at node n_{k_1+1} consumes at most w_R units of resource. Thus, L_2 can be obtained by concatenating some non-dominated label say L_1 corresponding to a path from the source to n_{k_1-1} generated during the forward recursion in $\overrightarrow{\mathcal{N}}$ with some non-dominated label say R_1 corresponding to a path from n_{k_1} to n_{i_1} generated during the backward recursion in $\overrightarrow{\mathcal{N}}$ rooted at n_{i_1} (see Figure 3.2). In other words, L_2 can be obtained by applying the procedure outlined in Section 3.3.1 treating n_{i_1} as the sink.

Given a label corresponding to a path from n_0 to n_{i_1} that dominates the label corresponding to the sub-path of P from n_0 to n_{i_1}, we then examine the next arc e_{k_2} ($k_2 > k_1$) for which $l_{e_{k_2}} < w_{e_{k_2}} < u_{e_{k_2}}$. Since k_2 is the first arc in P after k_1 where resource consumption is strictly between its lower and upper limits, we must have $w_j \in \{l_{e_j}, u_{e_j}\}$ for all $j = i_1 + 1, \ldots, k_2 - 1$. Thus, the label corresponding to the sub-path of P from n_0 to n_{k_2-1} is dominated by a label say L_3 corresponding to a path from n_0 to n_{k_2-1} generated by extending L_2 along arcs $e_{i_1+1}, \ldots, e_{k_2-1}$, and resource consumption at either their lower or upper limits. Furthermore, from Corollary 3.2.2 we again have that there exists $i_2 \geq k_2$ such that $\sum_{j=1,\ldots,i_2} w_j = U_{n_{i_2}}$

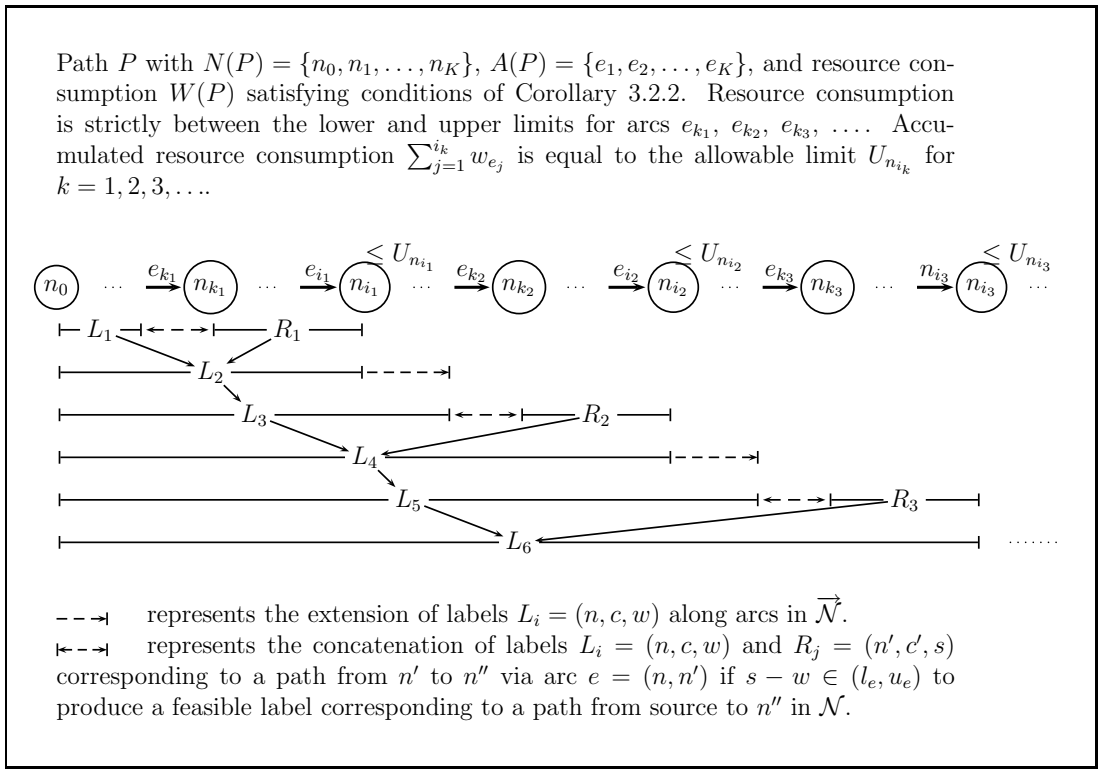


Figure 3.2: Constructing a path whose label dominates the label associated with a given path from source to sink and consumption profile satisfying Corollary 3.2.2 when multiple bounds may be imposed on accumulated resource consumption.

and $w_j \in \{l_{e_j}, u_{e_j}\}$ for all $j = k_2+1, \ldots, i_2$. Hence, the label corresponding to the sub-path of P from n_0 to n_{i_2} is dominated by some label say L_4 obtained by concatenating L_3 with some non-dominated label say R_2 corresponding to a path from n_{k_2} to n_{i_2} generated during the backward recursion in $\overrightarrow{\mathcal{N}}$ rooted at n_{i_2} (see Figure 3.2).

Continuing in the above fashion for each arc of P with resource consumption strictly between its lower and upper limits, we can construct a path whose associated label dominates the label corresponding to P. Thus, the path corresponding to an optimal solution from source to sink can be constructed by combining the non-dominated labels generated during backward recursion of the standard RCSPP in $\overrightarrow{\mathcal{N}}$ rooted at each node in the network that has a bound imposed and further combining these labels within a single forward pass of RCSPP in $\overrightarrow{\mathcal{N}}$ rooted at the source. Since the backward labels are generated through standard RCSPP techniques, we assume we have an efficient procedure to generate them and use these as an input to the forward recursion outlined in Algorithm 3. Here we define $\mathcal{R}_n$ to be the set of all non-dominated labels generated from the backward recursion rooted at node n.

As with standard DP approaches, at each iteration, the algorithm picks an unprocessed label and extends it with each out-going arc. However, in addition to extending along an arc with consumption at either the lower and upper limits of the arc, we also consider concatenation with labels in $\mathcal{R}_n$ for each $n \in N \setminus \{n_s\}$ that has a bound imposed. Two labels $L = (n, c, w)$ and $R = (n', c', s) \in \mathcal{R}_{n''}$ that are separated by an arc $e = (n, n')$ are concatenated if $l_e < s - w < u_e$. Concatenation results in a new label $L' = (n'', c + c' + \bar{c}_e(s - w), U_{n''})$ corresponding to a path from the source to n'' that consumes no more than $U_{n''}$ units of resource.

The **insert** procedure within Algorithm 3 refers to the standard process of checking newly generated labels for dominance against labels stored in $\mathcal{L}$, i.e., any newly generated label that is dominated by an existing label is immediately discarded otherwise the newly generated label is added to the list of non-dominated labels and

Input : $\mathcal{N} = (N, A)$ and $\mathcal{R}_n$ for each $n \in N \setminus \{n_s\}$
Initialize: $\mathcal{L} = \{\emptyset\}$;
mark $L_0 = (n_s, 0, 0)$ as unprocessed and **insert** into $\mathcal{L}$;
3.1 **while** $\exists$ *unprocessed labels in* $\mathcal{L}$ **do**
3.2 **pick** unprocessed label $L = (n, c, w)$ from $\mathcal{L}$;
3.3 `/* extend` L `along each outgoing arc */`
3.4 **forall** $e = (n, n') \in A$ **do**
3.5 `/* try extending with` l_e `*/`
3.6 **if** $w + l_e \leq U_{n'}$ **then**
3.7 mark $L' = (n', c + c_e + \bar{c}_e l_e, w + l_e)$ as unprocessed and **insert** into $\mathcal{L}$;
3.8 **end**
3.9 `/* try extending with` u_e `*/`
3.10 **if** $w + u_e \leq U_{n'}$ **then**
3.11 mark $L' = (n', c + c_e + \bar{c}_e u_e, w + u_e)$ as unprocessed and **insert** into $\mathcal{L}$;
3.12 **end**
3.13 `/* check for possibility to concatenate */`
3.14 **forall** $n'' \in N \setminus \{n_s\}$ *and* $R = (n', c', s) \in \mathcal{R}_{n''}$ *s.t.* $s - w \in (l_e, u_e)$ **do**
3.15 mark $L' = (n'', c + c' + c_e + \bar{c}_e(s - w), U_{n''})$ as unprocessed and **insert** into $\mathcal{L}$;
3.16 **end**
3.17 **end**
3.18 mark L as processed;
3.19 **end**
Output : $\arg\min\{c : L = (n_t, c, w) \in \mathcal{L}\}$

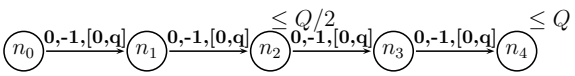
Algorithm 3: The forward DP recursion combining standard extensions along arcs in $\overrightarrow{\mathcal{N}}$ with non-dominated labels generated during the backward recursions.

any labels that are dominated by it are discarded. The reader is referred to Irnich and Desaulniers [77] for the various merits of invoking dominance procedures, and selecting unprocessed labels for extension (i.e., **pick** within Algorithm 3) that lead to label setting versus label correcting algorithms.

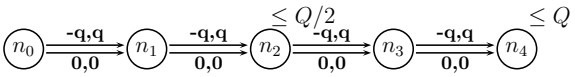
Algorithm 3 has at worst the same pseudo-polynomial state space as the naive approach of considering all integer points within the interval $[l_e, u_e]$ for resource consumption when extending a label along arc e. In addition, Algorithm 3 requires as an input, non-dominated labels generated by solving $|N| + 1$ standard RCSPPs.

Hence, at one extreme, if $l_e = u_e$ for all e, then we would have solved $|N| + 1$ standard RCSPPs when a single RCSPP would have sufficed. However, as the bounds on accumulated resource consumption and the number of integer values for resource consumption along arcs increase relative to the size of the network, the sufficient set of labels that need to be generated to ensure finding an optimal solution, and the total effort in generating these labels could be considerably less when fully exploiting the structure provided by Corollary 3.2.2. To see this, suppose we have some path P with $N(P) = \{n_0, n_1, n_2, n_3, n_4\}$ and are looking to generate the non-dominated labels associated with this path that are sufficient to ensure finding an optimal extension to the sink (see Figure 3.3.2 (a)). Here we assume $U_{n_4} = Q$ and $U_{n_2} = Q/2$, $l_e = 0$ and $u_e = q$ for all $e \in A(P)$ where $2q < Q < 3q$, and that $c_e = 0$ and $\bar{c}_e = -1$ for all $e \in A(P)$. Using the naive approach, there would be $Q + 1$ non-dominated labels associated with P (i.e., the labels $(n_4, -w, w)$ for $w = 0, \ldots, Q$). Furthermore, to obtain these labels, we would have generated a further 1, $Q/2+1$, and $Q/2+q+1$ non-dominated labels corresponding to paths ending at nodes n_0, n_1, n_2, and n_3 respectively. On the other hand, the backward recursion along P when only considering resource consumption at either l_e or u_e will lead to generating 3 non-dominated labels when rooted at node n_2 (see Figure 3.3.2 (d)), and 7 non-dominated labels when rooted at n_4 (see Figure 3.3.2 (c)). Furthermore, when combining these labels within a forward recursion rooted at n_0 in which we also only consider resource consumption at either l_e or u_e, we generate a further 17 non-dominated labels, 6 of which correspond to paths from n_0 to n_4 (see Figure 3.3.2 (e)).

Note that the number of labels generated when using the naive approach is as expected, pseudo-polynomial in q and Q. However in our case, the number of labels generated is constant and independent of Q and q. Of course, in general the effort in solving the forward and backward recursions when only considering extensions with l_e and u_e is in itself pseudo-polynomial. However, as the example demonstrates, if



3.3.2 (a): Network $\mathcal{N}$ induced by path from source n_0 to n_4 within a larger network. Along each arc e we display $c_e, \bar{c}_e, [l_e, u_e]$. Furthermore, $U_{n_4} = Q$, $U_{n_2} = Q/2$, $2q < Q < 3q$.



3.3.2 (b): Network $\overrightarrow{\mathcal{N}}$. Along each arc e we display c_e, w_e. As before, $U_{n_4} = Q$ and $U_{n_2} = Q/2$, and $2q < Q < 3q$.

3.3.2 (c): Labels generated during the backward recursion in $\overrightarrow{\mathcal{N}}$ rooted at node n_4:

Start node: n_0	n_1	n_2	n_3	n_4
	$R_8 = (n_1, -q, Q/2)$	$R_4 = (n_2, 0, Q/2)$	$R_2 = (n_3, 0, Q)$	$R_1 = (n_4, 0, Q)$
	$R_9 = (n_1, -2q, Q/2 - q)$	$R_5 = (n_2, -q, Q/2)$	$R_3 = (n_3, -q, Q - q)$	
	$R_{10} = (n_1, -2q, Q - 2q)$	$R_6 = (n_2, -q, Q/2)$		
		$R_7 = (n_2, -2q, Q - 2q)$		

3.3.2 (d): Labels generated during the backward recursion in $\overrightarrow{\mathcal{N}}$ rooted at node n_2:

Start node: n_0	n_1	n_2	n_3	n_4
	$R_{13} = (n_1, 0, Q/2)$	$R_{12} = (n_2, 0, Q/2)$		
	$R_{14} = (n_1, -q, Q/2 - q)$			

3.3.2 (e): Labels generated during the forward recursion in $\overrightarrow{\mathcal{N}}$ rooted at node n_0 together with the concatenation of non-dominated labels from the bacqward recursions:

End node: n_0	n_1	n_2	n_3	n_4
$L_1 = (n_0, 0, 0)$	$L_2 = (n_1, 0, 0)$	$L_5 = \langle L_1, R_{14} \rangle = (n_2, -Q/2, Q/2)$	$L_{13} = (n_3, -Q/2, Q/2)$	$L_4 = \langle L_1, R_{10} \rangle = (n_4, -Q, Q)$
	$L_3 = (n_1, -q, q)$	$L_6 = (n_2, 0, 0)$	$L_{14} = (n_3, -Q/2 - q, Q/2 + q)$	$L_{10} = \langle L_3, R_5 \rangle = (n_4, -Q/2 - q, Q)$
		$L_7 = (n_2, -q, q)$	$L_{16} = (n_3, 0, 0)$	$L_{15} = \langle L_5, R_3 \rangle = (n_4, -Q, Q)$
		$L_8 = (n_2, -q, q)$	$L_{17} = (n_3, -q, q)$	$L_{20} = \langle L_7, R_3 \rangle = (n_4, -Q, Q)$
		$L_{12} = \langle L_3, R_{12} \rangle = (n_2, -Q/2q, Q/2)$	$L_{18} = (n_3, -q, q)$	$L_{21} = (n_4, -Q/2, -Q/2)$
			$L_{19} = (n_3, -2q, 2q)$	$L_{22} = (n_4, -Q/2 - q, Q/2 + q)$
				$L_{23} = (n_4, -Q/2 - q, Q/2 + q)$
				$L_{24} = \langle L_{14}, R_1 \rangle = (n_4, -Q, Q)$
				$L_{25} = (n_4, 0, 0)$
				$L_{26} = (n_4, -q, q)$
				$L_{27} = (n_4, -q, q)$
				$L_{28} = (n_4, -2q, 2q)$
				$L_{29} = (n_4, -2q, 2q)$

○ ● – Label generated by extending arc in $\overrightarrow{\mathcal{N}}$.

□ ■ – Label generated by concatenation $\langle L, R \rangle$ where L is a label generated during the forward recursion and R is a label generated during a backward recursion.

● ■ – Dominated label.

Figure 3.3: The sufficient set of labels that need to be generated to ensure finding an optimal extension of a path.

the effort and the number of non-dominated labels generated when only considering extensions with l_e and u_e is significantly less than the naive approach, then significant savings can be achieved. In the case of FCSPP, it is easy to see how we can actually reach the worst case pseudo-polynomial bound when using the naive approach. In some cases, this may be unavoidable even when exploiting structure as we do. However, in many cases we can do significantly better than the naive approach by exploiting the fact that DP algorithms for RCSPP are typically much more efficient than their pseudo-polynomial worst case run time and/or state space suggest. We validate these observations empirically in our computational experiments later in Section 3.4.

3.3.3 Improving Efficiency of the DP Algorithm

So far we have not exploited the cost structure available in Proposition 3.2.1. Necessary conditions for the optimality of resource consumption along a path P where $A(P) = \{e_1, e_2, \ldots, e_K\}$, $N(P) = \{n_0, n_1, \ldots, n_K\}$ and consumption profile $W(P) = \{w_{e_1}, w_{e_2}, \ldots, w_{e_K}\}$ are:

1. $\bar{c}_{e_k} \leq \bar{c}_{e_j}$ for all $j \geq k$ such that $w_{e_k} > l_{e_k}$ and $w_{e_j} < u_{e_j}$, and
2. $\bar{c}_{e_k} \geq \bar{c}_{e_j}$ for all $j \geq k$ such that $w_{e_k} < u_{e_k}$, $w_{e_j} > l_{e_j}$, and $\sum_{i=1,\ldots,l} w_{e_i} < U_{n_l}$ for all $l = k, \ldots, j-1$.

Thus any label corresponding to paths that do not satisfy these conditions may be discarded during the labeling procedure even if they are non-dominated. Consider label L_7 in the example given in Figure 3.1 corresponding to the path (n_0, n_1, n_3) and resource consumption $(0, 3)$ on the respective arcs. Although L_7 is considered non-dominated with respect to the original dominance criteria, it does not satisfy conditions 1 and 2 and will never be part of an optimal solution. Indeed, the same path with resource consumption $(3, 0)$ has lower cost with the same amount of resource consumption. Of course, since in the second case the resource consumption is strictly

between its lower and upper limits for the first arc, this would not be recognized until we actually use L_7 in concatenation. Indeed, any path resulting from concatenating L_7 with another label would ultimately be dominated. However, by recognizing this earlier and eliminating L_7 we forgo any extension of L_7 and any concatenation of L_7 or concatenation of labels generated from L_7. Similarly, we can also discard labels L_{13} and L_{15}. Furthermore, we also do not need to consider the concatenation of labels corresponding to paths that individually satisfy these conditions but that would produce a label corresponding to a path that does not. For the example given in Figure 3.1, the only pair of labels that satisfy this condition after concatenation is the pair L_5 and R_5. These necessary conditions can be applied to any DP algorithm for FCSPP even the more naive approach or to enhance the dominance scheme proposed by Desaulniers [39] for SDVRPTW.

In addition to exploiting the cost structure, we can also be more efficient when concatenating labels. The process of concatenation (i.e., steps 3.14-3.16 within Algorithm 3) requires us to extend an unprocessed label $L = (n, c, w)$ corresponding to a path from the source to some node n along some arc $e = (n, n')$ and concatenate with all labels $R = (n', c', s)$ such that $s - w \in (l_e, u_e)$. Even if we group all non-dominated labels generated during the backward recursions by the start node of the corresponding paths, there may be as many as $U_{n'}|N|$ such labels within a group for node n' when clearly, there can be at most $(u_e - l_e + 1)|N|$ of these that can be feasibly concatenated with L. By sorting the labels generated during the backward recursion in increasing order of the additional amount of resource that each label can accommodate (for each start node) the set of labels that can lead to a feasible concatenation with L can be obtained quickly through binary search.

Finally, we can also restrict the extension of certain labels during the backward recursion to avoid duplicating effort in generating identical labels that can be obtained by different concatenations. Consider a label $R = (n_0, c', s)$ corresponding to path P

with $N(P) = \{n_0, \ldots, n_K\}$ in $\overrightarrow{\mathcal{N}}$ generated during the backward recursion rooted at n_K. If w is the amount of resource consumed along P and $U_{n_K} - w < s$, then there must exist $k < K$ where the amount w'' consumed along the arcs of P from n_0 to n_k is such that $s = U_{n_k} - w''$. In this case, any concatenation of R with some label L can also be obtained by concatenating L with the label R' corresponding to the sub-path of P from n_0 to n_k in $\overrightarrow{\mathcal{N}}$ and then extending the new label over the remaining arcs of P. Hence, we can stop any extension of a label during the backward recursion if the amount of additional resource it can accommodate is not determined by the bound on the node at which the search is rooted.

3.4 Computational Experiments

We test our algorithm on instances of FCSPP obtained from the pricing problem of the column generation formulations for three different classes of problems: SDVRPTW, PMSPCPT, and a multi-period IRP. All our algorithms were implemented in C, and all experiments were conducted on 2.4 GHz Dual AMD 250 processors with 4GB of RAM. We solve the root relaxation for each of these problems by solving each pricing iteration using our proposed DP algorithm (denoted by DP^1), and to have a comparison, either a naive approach (denoted by DP^3) or in the case of SDVRPTW, a standard DP procedure using our implementation of the dominance scheme proposed by Desaulniers [39] (denoted by DP^2). Of course in practice one generally uses a quick and efficient heuristic for generating columns and typically, only reverts to solving the pricing problem exactly when such heuristics fail. However, our goal is not to solve the root relaxations efficiently, but to solve FCSPP exactly (which is also necessary at some stage during column generation) and as efficiently as possible. To have as many exact pricing solves as possible, we attempt to solve each pricing iteration exactly and revert to heuristic methods only if both exact methods fail to terminate in the allocated time of one hour. Thus, we are using the master problem as a means of

generating different instances of FCSSP by providing different dual values and thus costs.

3.4.1 Experiments for SDVRPTW

For SDVRPTW, we use the well-known 56 benchmark instances from Solomon [100]. In keeping with the experimental setup of Desaulniers [39] and Gendreau et al. [65], we test the algorithms on each of the Solomon instances using vehicle capacity $Q = 30$, 50 and 100 units, as well as keeping only the first $n = 25$, 50 and 100 customers giving us a total of 504 instances. In addition to the fixed charge resource corresponding to vehicle capacity, we also need to keep track of time using standard DP techniques and apply dominance accordingly, i.e., in the forward recursion we keep track of the earliest time we can arrive at the end of the path, and in the backward recursion we keep track of the latest time we can arrive at the start of the path. Furthermore, for both DP^1 and DP^2, we eliminate cycles of size 4 or less by applying the dominance described in Irnich and Villeneuve [78].

Tables 3.1, 3.2, and 3.3 summarize the results for the three different classes of Solomon instances (C, R, and RC) corresponding to customer locations that are clustered, random, and both clustered and random. Each class is split into two subclasses corresponding to instances C1, R1, and RC1 with tighter restrictions on time windows and instances C2, R2, and RC2 with relatively loose restrictions on time windows. The first three columns of each table present the number of customers n, the vehicle capacity Q, and the total number of pricing iterations $Iter$ over all instances belonging to a given class, number of customers, and vehicle capacity. The next block of nine columns summarize results for those pricing iterations that could be solved by both our DP and a standard DP procedure using the dominance scheme proposed by Desaulniers [39] (i.e., DP^1 and DP^2) within the one hour time limit. We start with reporting the number of such pricing iterations It followed by label and timing

statistics for both approaches. Here B_1^1 and B_2^1 correspond to the average number of generated and non-dominated labels respectively per pricing iteration during the backward recursion of DP^1. Similarly, F_1^1 and F_2^1, and F_1^2 and F_2^2 represent the average number of generated and non-dominated labels per pricing iteration during the forward recursion of DP^1 and DP^2 respectively. t^1 and t^2 correspond to the average time (in seconds) to solve each pricing iteration for DP^1 and DP^2 respectively. Finally, the last block of 6 columns summarize the results for pricing iterations that could be solved only by DP^1. We start by reporting the number of such pricing iterations followed by the label and timing statistics. Note that it was never the case that DP^1 failed to solve a pricing iteration within the given time limit when DP^2 was successful.

In general, we observe that the clustered instances are the easiest to solve followed by the hybrid instances and then the random instances. As expected, the subclasses C2, R2, and RC2 are harder than their counterparts with tighter time windows, and the instances become harder to solve with increasing number of customers and/or vehicle capacity. We observe that we can solve within an hour a vast majority ($>$ 85%) of the pricing iterations using our approach. The exceptions include the RC2 instances with $n = 100$ and $Q = 100$, and the R2 instances with $n = 50$ and $Q = 100$, and $n = 100$ and $Q \geq 30$. For the $R2$ instances with $n = 100$, we can solve only 40%, 22%, and 9% of the pricing iterations for $Q = 30$, 50, and 100 respectively. The DP using Desaulniers' dominance scheme however begins to struggle to solve within an hour even some of the easier C2, and R1 instances with $n = 100$ and $Q \geq 50$, and the R2 and RC2 instances with $n \geq 50$ and $Q \geq 30$. As an aside, note that we can still solve the LP relaxations for almost all 504 instances. The exceptions include some of the R2 instances with $n = 100$ and $Q = 50$, and all but one of the R2 instances with $n = 100$ and $Q = 100$.

In terms of a time comparison for pricing iterations that could be solved by both approaches, we are on average 13 times faster over all instances. We are on average

as much as 175 times faster for the RC2 instances with $n = 50$ and $Q = 30$ and on average only 1.75 times faster for the C2 instances with $n = 100$ and $Q = 100$. In terms of the number of labels generated (i.e., $B_1^1 + F_1^1$ for DP^1 versus F_1^2 for DP^2) during pricing iterations solved by both approaches, we observe that we generate on average 1.5 to 7 times fewer labels. We also observe that we generate on average 1 to 4 times fewer non-dominated labels (i.e., $B_2^1 + F_2^1$ for DP^1 versus F_2^2 for DP^2). Note that not only do we generate and store fewer labels, but that the effort in managing these labels is considerably less since we do not need to check dominance of labels generated during the forward recursion against those generated in the backward recursion and vice versa. Thus, the computational effort can be significantly less as reflected in the run times.

In summary, we outperform the DP using Desaulniers' dominance scheme on all instances, although the degree varies by class and difficulty and tends to diminish as the problems become more difficult. Indeed, as the problems become harder (i.e., as number of customers, capacity, and time-window increase) the state space is ultimately dominated by labels required to prevent cycles of size 4 or less, which is the same mechanism implemented in both DP^1 and DP^2. Nonetheless, the value of a decomposition approach over a standard DP algorithm that relies on a dominance scheme that augments the state space is still evident.

3.4.2 Experiments for PMSPCPT

For PMSPCPT, we test our algorithm on instances derived from Chen and Powell [24] for the traditional parallel machine scheduling problem with due dates and minimizing weighted number of tardy jobs by not allowing jobs to be tardy, but instead by allowing the processing times to be reduced at a cost. We test our algorithm against the naive approach for instances with number of machines $m = 2$, 4, 6, 8, and 10, number of jobs $n = 20$, 40, 60, and 80, job processing times p_j an integer drawn

uniformly within the interval $[1, 100]$, and job due dates d_j to be the maximum of p_j and an integer drawn uniformly within the interval $[1, 100n/(mq)]$ where $q \in \{1, 2, 3\}$ indicates the level of congestion of the scheduling system. The per unit cost $\bar{c}_j$ of lowering processing time below p_j is chosen uniformly within the interval $(0, 1]$. Note that for this class of problems, we initiate a backward recursion rooted at each node in the network since the number of arcs in the optimal solution with resource consumption strictly between its lower and upper limit is not known beforehand. Furthermore, since we do not have release dates for jobs, we may assume an ordering of jobs by due date (see [24]) and thus the network we use for pricing is acyclic.

Tables 3.4 and 3.5 summarize the results for these instances. For a given number of jobs n, number of machines m, and congestion factor q, the tables display the total number of pricing iterations $Iter$ followed by label and timing statistics for both DP^1 and DP^3. As before, B_1^1 and B_2^1 correspond to the average number of generated and non-dominated labels respectively per pricing iteration during the backward recursions of DP^1, and F_1^1 and F_2^1, and F_1^3 and F_2^3 represent the average number of generated and non-dominated labels per pricing iteration during the forward recursion of DP^1 and DP^3 respectively. Similarly, t^1 and t^3 correspond to the average time (in seconds) to solve each pricing iteration for DP^1 and DP^2 respectively. Note that none the pricing iterations exceeded the one hour time limit for both DP^1 and DP^3.

As expected, the problems get harder to solve with more jobs and/or fewer machines. More interestingly, the problems become considerably harder as we increase congestion factor from 1 to 2 but only marginally more difficult when increasing congestion factor from 2 to 3. Indeed, as we increase the congestion factor, many more jobs are required to be processed below their nominal processing times. Since, lowering processing times is reflected in an increase in costs, the opportunities for dominance among possible labels weakens. This is reflected in the large jump in

number of labels being generated and stored for both DP^1 and DP^3 when increasing the congestion factor from 1 to 2. However, as the congestion factor is further increased to 3, the number of jobs with processing times below their nominal values does not increase as much. As a result, the increase in number of labels generated and stored is comparatively small.

In terms of a time comparison between DP^1 and DP^3, we are on average 123 times faster than the naive approach over all instances. As expected, our performance advantage is at its greatest when the congestion factor is the lowest, i.e. when the bound on accumulated resource consumption is the largest. We are on average 269 times faster when $q = 1$ as compared to around 50 times faster when $q \geq 2$. Similarly, we observe that we generate on average 80 times fewer labels than the naive approach (175 times fewer when $q = 1$ and about 32 times fewer when $q \geq 2$). Note, that B_1^1 is tallied over all backward recursions (one for each node). Thus, since we do not check dominance against paths that start and end at different nodes, generating fewer overall labels has an even greater effect on the total effort reflected in the run time.

3.4.3 Experiments for a Multi-Period IRP

For IRP, we test our algorithm on instances belonging to a multi-period variant motivated by practical problems in maritime transportation. These instances are characterized by the number of periods T in the planning horizon, the number of supply and demand locations n_S and n_D respectively, vehicle capacity Q, and f_j for $j = 1, \ldots, n_S + n_D$ the maximum amount of inventory that can be loaded/unloaded at a supply/demand point during a single period. Here, we use a time-expanded network to represent a vehicle's route in which nodes correspond to a particular time period as well as a particular location. The amount of resource consumed when traversing an arc is bounded between 0 and f_j if the head of the arc corresponds to a supply facility at j or between $-f_j$ and 0 if it corresponds to a demand facility at j. The

total amount of resource consumption accumulated up to a node is bounded below by 0 and bounded above by Q except for the sink node where it is bounded above by 0.

We test our algorithm against the naive approach for instances with number of time periods $T = 60$ in which the number of periods taken to travel between facilities is given by randomly locating supply/demand points within a 12×12 grid. Instances include number of supply and demand facilities n_S, $n_D = 3$, 4, 5, 6, 7 and 8, vehicle capacity $Q = 30$, 50 and 100 units, and f_j an integer drawn uniformly from $[1, 2Q/q]$ where $q \in \{1, 2, 3\}$. Thus, $1/q$ corresponds to the rate relative to vehicle capacity of inventory that can be loaded/unloaded during a single time period. Production and consumption rates, as well as storage capacities at supply/demand points are chosen to ensure relocation of inventory is necessary in a feasible solution. Additionally, the fixed transportation costs and profit from supplying inventory to demand facilities is chosen to encourage transporting inventory from suppliers to consumers. Since the master problem contains the facility inventory variables and lot-sizing constraints that ensure that a supply facility never exceeds its storage capacity or that inventory is never depleted at a demand facility, these characteristics (i.e., production/consumption rates, storage capacities at facilities, and profit from supplying demand) do not explicitly feature in the pricing problem although they ensure that the solutions to the resulting pricing problems are non-trivial.

Since we have both upper and lower bounds on the consumption of the resource corresponding to vehicle capacity, and this resource can be both consumed and replenished, we need to modify the dominance criteria to check resource consumption with equality during the forward recursion, and include the minimum amount of additional resource that needs to be consumed in order to meet some lower bound on resource consumption accumulated along a path as part of the dominance criteria during the backward recursion. In addition, we also need to modify the concatenation process so

that when concatenating labels separated by an arc, the resource consumption along the arc is chosen to try and make up the difference between the resource consumed within the forward label and both, the maximum amount of resource that can be additionally accommodated in the backward label as well as the minimum amount of resource that is necessary to meet all lower bounds.

Tables 3.6 and 3.7 summarize the results for these instances in the same format as before. We observe that the problems get harder to solve with more facilities. Furthermore, although the problems tend to get harder to solve using DP^1 as we increase q (i.e., decrease the amount that can be loaded/unloaded during a single time period), we notice the opposite effect for DP^3. For the same number of facilities and vehicle capacity, increasing q decreases the number of integer values for resource consumption along an arc making the naive approach more tractable. On the other hand, the optimal paths tend to get longer since more visits to facilities are required to load/unload the same amount of inventory. In the naive approach we may generate (and discard) such paths even when $q = 1$. However, in our approach, since arcs have resource consumption satisfying Corollary 3.2.2, we tend to only generate relatively small paths when $q = 1$. Thus increasing q can have much more of a detrimental impact on the efficiency of our approach as compared to the naive approach.

In terms of a time comparison between DP^1 and DP^3, we are on average 9 times faster than the naive approach over all instances. The performance advantage is at its greatest when q is as small as possible, i.e., when the number of possible integer values for resource consumption along an arc is the largest. We are on average 21 times faster when $q = 1$ as compared to around 4 and 2 times faster when $q = 2$ and 3 respectively. We observe a similar trend for number of labels generated. Finally, note that each time we add a pair of facilities, we increase the number of nodes in the network by $2T$ requiring an additional $2T$ backward recursions. Thus, for the same Q and q, the gains from our approach diminish quickly with increasing n_S and/or n_D.

3.5 Conclusions

Branch-and-price algorithms are among the most successful exact optimization approaches for solving many routing and scheduling problems. This is due, in part, to the availability of extremely efficient and effective dynamic programming algorithms for solving resource constrained shortest path problems. Unfortunately, in many situations, incorporating relevant practical constraints results in pricing problems that do not give rise to pure resource constrained shortest path problems, but more complex variants, such as the fixed charge shortest path problem. We have shown that by exploiting the structure of optimal solutions to the fixed charge shortest path problem, it is possible to design and implement a solution approach that relies on solving multiple resource constrained shortest path problems, and therefore can again make use of these extremely efficient and effective dynamic programming algorithms. This extends the class of routing and scheduling problems that can be successfully solved using branch-and-price algorithms.

Table 3.1: Results from experiments on the clustered C1 and C2 class of the Solomon instances for SDVRPTW.

(a) C1 instances

			Pricing iterations solved by DP^1 and DP^2									Pricing iterations solved only by DP^1					
				DP^1					DP^2				DP^1				
n	Q	$Iter$	It	B_1^1	B_2^1	F_1^1	F_2^1	t^1 (s)	F_1^2	F_2^2	t^2 (s)	It	B_1^1	B_2^1	F_1^1	F_2^1	t^1 (s)
				$\times 10^3$					$\times 10^3$				$\times 10^3$				
25	30	114	114	0.4	0.1	3.6	0.9	0.03	17	2.3	0.6	0	-	-	-	-	-
25	50	133	133	3.1	0.9	14	1.8	0.6	61	4.7	6.9	0	-	-	-	-	-
25	100	208	207	45	3.6	29	2.1	2.6	176	8.2	27	1	966	90	1,418	98	1,461
50	30	171	165	2.2	0.4	27	7.6	3.1	106	8.3	37	6	11	4.6	349	144	240
50	50	244	244	18	2.6	69	3.9	3.7	264	8.4	25	0	-	-	-	-	-
50	100	363	359	98	5.9	60	2.5	6.9	491	11	43	3	3,260	139	2,612	96	2,065
100	30	248	214	7.9	0.6	63	3.7	1.4	291	8.7	19	34	38	10	746	174	611
100	50	371	325	74	5.7	237	7.3	23	951	18	141	44	174	28	1,953	431	1,503
100	100	627	539	189	3.6	363	8.2	26	1,329	21	115	82	137	37	2,883	722	2,367

(b) C2 instances

			Pricing iterations solved by DP^1 and DP^2									Pricing iterations solved only by DP^1					
				DP^1					DP^2				DP^1				
n	Q	$Iter$	It	B_1^1	B_2^1	F_1^1	F_2^1	t^1 (s)	F_1^2	F_2^2	t^2 (s)	It	B_1^1	B_2^1	F_1^1	F_2^1	t^1 (s)
				$\times 10^3$					$\times 10^3$				$\times 10^3$				
25	30	96	96	0.5	0.1	4.7	1.5	0.1	34	5.4	3.5	0	-	-	-	-	-
25	50	134	134	3.2	0.8	30	4.6	3.0	156	12	46	0	-	-	-	-	-
25	100	226	193	27	2.3	50	3.4	2.1	190	10	19	32	343	28	1,695	96	701
50	30	169	142	2.3	0.3	20	2.6	0.3	116	6.6	6.7	27	8.1	2.6	241	137	160
50	50	219	190	21	2.7	96	5.0	5.5	417	13	47	23	29	9.1	855	353	567
50	100	350	304	370	13	266	8.2	31	1,038	24	99	30	3,220	127	4,847	293	2,851
100	30	236	197	10	0.8	106	11	5.1	803	27	99	36	38	6.0	965	250	804
100	50	361	285	81	4.9	489	13	33	2,254	39	358	44	1,118	93	6,872	106	1,321
100	100	566	378	681	17	938	14	151	3,535	39	263	127	3,208	151	11,882	140	2,623

Table 3.2: Results from experiments on the random R1 and R2 class of the Solomon instances for SDVRPTW.

(a) R1 instances

			Pricing iterations solved by DP^1 and DP^2									Pricing iterations solved only by DP^1					
				DP^1					DP^2				DP^1				
n	Q	$Iter$	It	B_1^1	B_2^1	F_1^1	F_2^1	t^1 (s)	F_1^2	F_2^2	t^2 (s)	It	B_1^1	B_2^1	F_1^1	F_2^1	t^1 (s)
				$\times 10^3$					$\times 10^3$				$\times 10^3$				
25	30	206	206	0.9	0.3	11	1.5	0.1	32	3.5	0.8	0	-	-	-	-	-
25	50	253	253	5.1	1.1	19	1.9	0.2	60	4.6	2.0	0	-	-	-	-	-
25	100	287	287	22	2.1	16	1.5	0.7	53	3.6	1.4	0	-	-	-	-	-
50	30	319	319	8.1	1.8	162	13	17	493	28	115	0	-	-	-	-	-
50	50	382	362	80	9.3	397	17	54	1,237	41	342	19	748	71	2,889	113	929
50	100	546	455	288	13	218	8.5	59	744	21	134	46	5,494	207	3,363	103	2,043
100	30	562	551	74	8.6	1,430	46	115	4,205	114	771	11	217	28	6,332	249	1,362
100	50	608	448	398	22	1,652	37	208	4,494	76	852	144	1,796	99	7,680	180	1,322
100	100	1,090	723	1,561	36	1,516	29	285	6,913	106	1,249	196	13,060	299	14,610	268	2,933

(b) R2 instances

			Pricing iterations solved by DP^1 and DP^2									Pricing iterations solved only by DP^1					
				DP^1					DP^2				DP^1				
n	Q	$Iter$	It	B_1^1	B_2^1	F_1^1	F_2^1	t^1 (s)	F_1^2	F_2^2	t^2 (s)	It	B_1^1	B_2^1	F_1^1	F_2^1	t^1 (s)
				$\times 10^3$					$\times 10^3$				$\times 10^3$				
25	30	197	197	1.5	0.6	55	10	6.7	201	25	63	0	-	-	-	-	-
25	50	255	235	17	3.9	238	19	48	797	52	446	18	100	22	1,396	103	724
25	100	353	226	153	11	335	18	62	1,155	47	394	95	1,278	87	3,527	174	1,865
50	30	309	204	8.9	1.9	315	28	65	813	53	233	92	31	8.4	2,102	229	1,462
50	50	374	131	52	5.0	472	22	43	1,568	57	369	215	508	54	4,840	189	1,477
50	100	664	134	315	11	865	23	55	3,065	62	408	304	1,767	147	10,463	331	2,719
100	30	470	131	39	3.4	1,240	46	130	3,969	121	1,106	56	262	20	4,991	173	1,568
100	50	702	57	77	2.8	455	8.5	3.2	1,454	23	30	96	532	22	9,542	190	2,124
100	100	782	69	291	6.0	961	13	10	3,239	36	81	0	-	-	-	-	-

Table 3.3: Results from experiments on the hybrid RC1 and RC2 class of the Solomon instances for SDVRPTW.

(a) RC1 instances

			Pricing iterations solved by DP^1 and DP^2									Pricing iterations solved only by DP^1					
				DP^1					DP^2				DP^1				
n	Q	$Iter$	It	B_1^1	B_2^1	F_1^1	F_2^1	t^1 (s)	F_1^2	F_2^2	t^2 (s)	It	B_1^1	B_2^1	F_1^1	F_2^1	t^1 (s)
				$\times 10^3$					$\times 10^3$				$\times 10^3$				
25	30	98	98	0.2	0.1	2.6	0.6	0.005	8.6	1.5	0.05	0	-	-	-	-	-
25	50	112	112	1.1	0.3	6.8	1.3	0.03	24	2.7	0.4	0	-	-	-	-	-
25	100	137	137	11	1.8	19	2.2	0.3	62	4.8	1.7	0	-	-	-	-	-
50	30	145	145	1.4	0.3	14	2.1	0.1	56	5.5	1.6	0	-	-	-	-	-
50	50	183	183	10	1.8	46	4.7	1.3	176	11	12	0	-	-	-	-	-
50	100	333	333	188	13	163	10	22	526	24	53	0	-	-	-	-	-
100	30	274	272	15	2.9	635	26	58	1,647	60	328	2	118	24	4,348	149	777
100	50	331	270	145	12	937	26	85	2,777	64	523	49	1,259	113	8,093	178	1,852
100	100	487	355	748	23	703	19	80	2,291	47	182	88	4,464	148	7,898	168	1,813

(b) RC2 instances

			Pricing iterations solved by DP^1 and DP^2									Pricing iterations solved only by DP^1					
				DP^1					DP^2				DP^1				
n	Q	$Iter$	It	B_1^1	B_2^1	F_1^1	F_2^1	t^1 (s)	F_1^2	F_2^2	t^2 (s)	It	B_1^1	B_2^1	F_1^1	F_2^1	t^1 (s)
				$\times 10^3$					$\times 10^3$				$\times 10^3$				
25	30	97	97	0.2	0.1	5.0	1.5	0.03	27	6.3	1.6	0	-	-	-	-	-
25	50	120	120	1.6	0.5	26	6.5	2.6	154	19	47	0	-	-	-	-	-
25	100	148	127	34	4.4	187	14	32	679	35	282	20	313	52	1,599	119	1,182
50	30	142	135	1.8	0.4	38	11	2.7	512	73	476	7	5.6	2.2	242	110	106
50	50	217	151	14	2.2	172	13	14	840	35	191	54	46	16	1,620	221	1,287
50	100	362	239	388	19	538	17	101	1,869	41	319	79	398	89	7,820	260	2,583
100	30	319	141	14	2.0	929	24	41	2,631	64	303	114	63	14	7,240	222	2,171
100	50	316	121	162	9.2	1,886	28	96	5,598	74	604	62	260	32	8,125	170	1,575
100	100	433	167	1,041	22	3,274	42	219	10,876	109	1,127	28	3,153	60	9,348	108	1,279

Table 3.4: Results from experiments on the instances of PMSPCPT with $n = 20$ and 40.

n	m	q	$Iter$	DP^1					DP^3		
				B_1^1	B_2^1	F_1^1	F_2^1	t^1 (s)	F_1^3	F_2^3	t^3 (s)
				$\times 10^3$					$\times 10^3$		
20	2	1	223	0.1	0.1	2.6	0.2	0.002	881	1,963	2.0
		2	274	6.4	1.4	24	0.8	0.115	1,952	3,523	3.5
		3	312	7.1	1.5	23	0.7	0.048	2,131	3,209	3.2
20	4	1	144	0.5	0.2	2.0	0.1	0.002	731	1,700	1.7
		2	216	3.4	0.9	10	0.4	0.012	1,145	2,288	2.3
		3	212	2.4	0.6	9.6	0.3	0.008	1,051	1,854	1.9
20	6	1	117	0.9	0.3	2.2	0.1	0.002	764	1,677	1.7
		2	204	2.0	0.6	7.0	0.2	0.006	870	1,779	1.8
		3	213	1.3	0.3	6.8	0.2	0.005	733	1,448	1.4
20	8	1	120	1.2	0.3	2.2	0.1	0.002	733	1,586	1.6
		2	186	1.3	0.4	5.3	0.2	0.004	652	1,496	1.5
		3	200	0.9	0.2	4.9	0.2	0.003	609	1,253	1.3
20	10	1	119	1.2	0.3	2.2	0.1	0.002	691	1,531	1.5
		2	170	0.9	0.3	4.0	0.2	0.003	560	1,311	1.3
		3	188	0.9	0.2	4.1	0.2	0.003	578	1,182	1.2
40	2	1	607	0.5	0.3	41	1.0	0.066	8,120	7,050	7.0
		2	627	223	21	1,023	5.6	13.92	24,220	12,786	13
		3	642	305	22	986	4.6	8.00	26,107	10,504	11
40	4	1	286	2.7	0.8	20	0.5	0.03	5,132	4,376	4.4
		2	427	128	13	436	2.2	1.74	11,571	6,819	6.8
		3	460	106	9.6	426	1.8	0.92	11,125	5,497	5.5
40	6	1	246	5.8	1.2	17	0.4	0.02	4,440	3,935	3.9
		2	394	70	7.7	287	1.4	0.52	8,534	5,103	5.1
		3	450	45	4.7	229	1.1	0.26	7,046	4,090	4.1
40	8	1	256	10	1.8	20	0.4	0.03	4,099	3,798	3.8
		2	407	40	5.2	192	1.0	0.24	6,382	4,273	4.3
		3	419	22	2.5	144	0.8	0.13	5,052	3,370	3.4
40	10	1	243	12	1.7	19	0.3	0.02	3,926	3,645	3.6
		2	387	23	3.0	112	0.7	0.10	4,896	3,687	3.7
		3	433	16	1.7	89	0.7	0.08	3,993	2,997	3.0

Table 3.5: Results from experiments on the instances of PMSPCPT with $n = 60$ and 80.

n	m	q	$Iter$	DP^1					DP^3		
				B_1^1	B_2^1	F_1^1	F_2^1	t^1 (s)	F_1^3	F_2^3	t^3 (s)
				$\times 10^3$					$\times 10^3$		
60	2	1	821	0.6	0.3	112	2.1	0.53	21,316	13	130
		2	917	1,140	60	6,241	12	224.21	76,098	25	1,183
		3	798	2,331	71	10,088	10	198.28	87,737	20	1,412
60	4	1	475	5.8	1.3	64	1.0	0.29	12,453	7.4	46
		2	668	1,449	65	6,485	6.0	66.77	37,261	14	238
		3	663	1,747	56	8,955	5.5	49.89	41,742	11	251
60	6	1	415	14	2.3	50	0.7	0.16	11,152	6.7	35
		2	603	887	44	3,466	3.6	16.66	24,534	10	85
		3	658	1,123	39	4,817	3.9	16.12	28,982	8.6	99
60	8	1	361	27	3.5	53	0.7	0.12	10,468	6.2	25
		2	620	555	31	2,238	2.7	7.11	16,890	8.1	43
		3	627	589	25	2,977	2.9	6.23	22,217	7.0	50
60	10	1	361	37	4.1	58	0.6	0.13	10,018	5.9	21
		2	612	500	29	2,102	2.8	4.08	16,447	7.8	34
		3	591	270	14	1,682	2.0	2.33	16,142	5.9	27
80	2	1	1421	0.7	0.4	327	4.6	4.00	36,019	18	334
		2	1201	380	25	1,906	10	48.98	106,384	31	1,752
		3	1316	3,201	86	9,402	12	268.91	145,415	28	3,236
80	4	1	569	8.2	1.9	168	1.8	2.89	23,799	11	114
		2	788	5,550	153	16,895	10	603.67	81,713	23	846
		3	704	9,407	164	30,237	10	555.76	103,250	19	1,188
80	6	1	477	27	4.6	109	1.2	0.83	18,648	8.5	59
		2	682	3,905	116	12,872	7.2	288.84	56,448	17	354
		3	870	6,981	137	25,053	8.2	221.33	69,389	15	437
80	8	1	411	48	6.0	105	1.0	0.60	18,090	8.3	52
		2	745	2,756	93	8,688	5.5	81.69	42,480	14	166
		3	880	4,123	92	16,079	5.7	75.66	48,791	12	193
80	10	1	379	84	7.7	120	1.0	0.54	18,374	8.5	46
		2	766	1,985	76	7,105	4.7	34.07	32,472	12	98
		3	871	2,291	65	11,797	4.4	32.72	39,615	10	112

Table 3.6: Results from experiments on the instances of a multi-period IRP with $n_S, n_D = 3, 4$ and 5.

$n_S \times n_D$	Q	q	$Iter$	DP^1					DP^3		
				B_1^1	B_2^1	F_1^1	F_2^1	t^1 (s)	F_1^3	F_2^3	t^3 (s)
				$\times 10^3$					$\times 10^3$		
3x3	30	1	314	28	9	34	1.5	0.1	346	6	0.4
		2	259	51	15	66	2.8	0.1	260	6	0.3
		3	224	56	15	82	3.2	0.1	224	6	0.3
3x3	50	1	354	29	8	41	1.8	0.1	905	10	1.6
		2	287	68	19	100	3.8	0.2	631	9	1.1
		3	233	91	23	162	5.7	0.3	530	10	0.9
3x3	100	1	388	32	9	58	2.7	0.1	3,383	21	11
		2	279	82	22	141	6.0	0.3	2,311	19	7.3
		3	239	136	31	305	9.1	0.7	1,837	19	5.9
4x4	30	1	401	81	20	102	2.7	0.2	618	8	0.8
		2	302	157	37	204	4.3	0.4	482	8	0.6
		3	281	188	41	261	5.0	0.5	412	8	0.5
4x4	50	1	487	84	20	128	3.4	0.2	1,600	14	2.9
		2	330	221	50	343	7.0	0.7	1,193	13	2.2
		3	283	274	57	464	8.6	1.0	987	13	1.8
4x4	100	1	477	104	23	201	4.9	0.4	6,044	28	20
		2	327	267	58	480	9.8	1.1	4,321	26	15
		3	275	425	84	888	14	2.4	3,462	26	12
5x5	30	1	453	151	33	186	3.5	0.4	1,010	11	1.3
		2	345	293	60	397	5.6	0.9	765	10	1.0
		3	311	392	74	554	6.6	1.3	651	9	0.8
5x5	50	1	503	155	33	211	4.2	0.4	2,590	17	4.6
		2	375	404	79	650	9.2	1.6	1,890	16	3.4
		3	318	542	98	949	11	2.3	1,566	16	2.8
5x5	100	1	583	183	38	284	6.1	0.6	9,810	35	33
		2	373	545	103	1,041	15	2.8	7,017	32	24
		3	331	846	148	1,848	19	5.8	5,583	32	19

Table 3.7: Results from experiments on the instances of a multi-period IRP with $n_S, n_D = 6, 7$ and 8.

$n_S \times n_D$	Q	q	$Iter$	DP^1					DP^3		
				B_1^1	B_2^1	F_1^1	F_2^1	t^1 (s)	F_1^3	F_2^3	t^3 (s)
				$\times 10^3$					$\times 10^3$		
6x6	30	1	508	274	53	331	4.6	0.8	1,473	13	1.9
		2	349	486	89	670	7.1	1.8	1,108	11	1.5
		3	347	616	105	911	8.1	2.4	954	11	1.3
6x6	50	1	610	284	53	405	6.1	0.9	3,777	21	6.8
		2	401	678	120	1,121	12	3.1	2,789	19	5.0
		3	336	903	148	1,637	13	4.9	2,278	19	4.2
6x6	100	1	830	223	41	386	5.3	0.9	8,711	26	30
		2	1144	257	44	571	5.4	1.8	2,849	11	9.8
		3	1396	260	41	611	4.1	2.2	1,467	7	4.9
7x7	30	1	534	512	88	621	5.9	1.9	1,955	15	2.8
		2	406	928	151	1,272	8.4	4.3	1,445	13	2.1
		3	383	1,044	161	1,553	9.3	5.2	1,216	13	1.8
7x7	50	1	575	544	91	760	8.0	2.2	4,963	25	9.9
		2	396	1,357	215	2,145	13	8.7	3,507	22	7.0
		3	376	1,559	230	2,814	15	11	2,904	22	6.0
7x7	100	1	642	681	113	1,090	13	3.7	18,959	50	79
		2	440	1,707	265	3,462	23	15	13,174	44	55
		3	331	2,546	362	5,831	26	29	10,324	44	44
8x8	30	1	664	1,147	170	1,763	16	6.3	25,083	56	89
		2	505	2,855	398	5,955	29	30	17,930	51	62
		3	370	4,096	517	10,095	31	54	14,253	50	50
8x8	50	1	563	921	140	1,102	6.9	3.9	2,556	17	3.5
		2	478	1,441	210	2,001	9.9	7.9	1,913	15	2.6
		3	367	1,744	238	2,729	11	11	1,636	15	2.2
8x8	100	1	653	991	149	1,354	10	4.6	6,583	28	12
		2	424	2,205	314	3,577	16	18	4,793	25	8.8
		3	417	2,661	350	5,227	18	23	3,938	24	7.4

CHAPTER IV

A BRANCH-PRICE-AND-CUT ALGORITHM FOR A MARITIME INVENTORY ROUTING PROBLEM

4.1 Introduction

The *inventory routing problem* (IRP) is concerned with the distribution of a product from supply facilities to demand facilities over a given planning horizon using a set of vehicles. The objective is to minimize costs and/or maximize revenue related to distributing the product without depleting inventory and/or exceeding storage capacity at each of the facilities. IRP is at the heart of many supply chain problems, none more complex and intricate than managing the sourcing, manufacturing, and distribution of various products within the petrochemical industry. In 2005 2.4 billion tons (17.6 billion barrels) of petroleum was shipped by maritime transportation [93]. This accounts for approximately 62% of all petroleum produced during this year and approximately $182 billion in trade value of crude oil imported to the US alone [53]. In addition to the huge capital investment and large operational costs required to keep the supply chain moving, the sheer volume and value of product passing through the supply chain makes this a particularly enticing problem for optimization.

Its not surprising that combining routing and inventory management makes IRP an extremely hard problem to solve. The example illustrated in Bell et al. [13] and reexamined by Adelman [1] and Song and Savelsbergh [102] demonstrates the difficulty involved with finding an optimal solution to a relatively small problem. The problem is nontrivial even in the presence of only two customers as demonstrated by the example given by Campbell et al. [21]. Understandably, IRP has not received the same level of attention as pure routing or scheduling problems, or pure inventory

management problems. Nonetheless, there still exists a reasonable volume of work dedicated to this area. We refer the reader to Bertazzi et al. [17], Cordeau et al. [34], Campbell et al. [22], and Campbell et al. [21] for an overview of existing solution approaches and models for IRP including stochastic variants and asymptotic analysis.

Our focus is in solving the tactical level deterministic IRP motivated by maritime transportation. Unfortunately, in addition to assumptions related to having a single (often uncapacitated) supply facility, constant production/demand rates, constant port capacities, a homogeneous fleet, and short planning horizons, deterministic models for IRP often make further assumptions about the routing component that are either impractical or simply incorrect for maritime transportation.

In some of the earliest work on IRP, Federgruen and Zipkin [56] consider a single period IRP which is essentially a vehicle routing problem in which the amount of a customer's demand that is to be satisfied is itself a continuous bounded variable subject to holding and stock-out costs. A more practical multi-period problem is the one considered by Fisher et al. [59] and Bell et al. [13]. In their solution approach, they precompute a set of possible routes servicing customers and explicitly include these routes within an integer programming model in which decisions regarding the routes to be used, their respective start times and supply quantities to customers visited on the respective routes are determined. In addition to assuming that it is possible to explicitly include a good representative set of all possible routes, their approach assumes that a route can be defined by a sequence of customers and a start time. In maritime transportation it is often the case that a vessel may have to remain demurraged (idle) at a port waiting for sufficient inventory to accumulate at a supply port before loading, or sufficient inventory to be consumed at a demand port before discharging a product. Furthermore, due to large capacities, vessels may load/discharge at a port several times on its route. Hence, a vessel acts as both a mode of transportation and as temporary storage for inventory in transit. Thus,

the vessel incurs transportation costs and idle time costs. As a result, the timing and frequency of individual ports visited becomes an essential component in defining the route. The possible number of such routes within a planning horizon spanning several months makes explicitly including them within a model virtually impossible. Although several heuristic methods have since been developed to solve larger more complex problems than the one introduced by Bell et al. [13], the prevalent model for deterministic IRP has remained relatively unchanged.

Campbell et al. [23] use a similar time indexed formulation to Bell et al. However, rather than precomputing the routes a priori, they propose a two phase rolling horizon approach in which the timing and suggested delivery quantities are determined in the first phase, and the routes and actual delivery quantities are determined in a second phase by essentially solving a sequence of vehicle routing problems. Their model assumes that the routes servicing a set of customers can be completed within a single time period. This allows for a certain level of decoupling of routing decisions since there are no spacial or temporal constraints coupling routing decisions from one time period to another. Although this justifies their two-phase approach and the use of insertion heuristics for constructing routes over a rolling horizon, it is certainly not a practical assumption for maritime transportation in which a single voyage between two ports can span several time periods. The work of Bard et al. [8] is along similar lines in terms of using a two phase rolling horizon approach in which delivery quantities are determined first and the delivery routes are constructed by solving a vehicle routing problem for each time period.

In addition to considering multiple supply facilities, the work of Savelsbergh and Song [99] on IRP with continuous moves captures a richer routing component by considering routes that may span several time periods and that may also have idle time between visits to customers or a supply facility. Their routing decisions are included in a time indexed multicommodity flow formulation. Despite tightening

their formulation through network preprocessing and constraints specifically designed to cut-off fractional solutions, the size of problems that can be solved are relatively small spanning only a few days of the time horizon. However, their multicommodity flow formulation does serve to solve much larger problems when embedded in a large neighborhood search heuristic.

In contrast to the aforementioned examples, the work by Christiansen and Nygreen [28, 29], Christiansen [26], and Christiansen et al. [27] on IRP specifically for maritime transportation addresses some of the shortcomings of the routing component within the more traditional IRP models just mentioned. To highlight the intrinsic difficulties for IRP in maritime transportation, Christiansen [26] makes a point to distinguish between distribution problems and inventory pickup-and-delivery problems where there is less of an asymmetry between the number of suppliers and customers, routes may span a significant portion of the time horizon, and idle time may be necessary between loads/discharges at ports. They use a continuous time column generation formulation in which decision epochs are indexed by the sequence in which vessels load/discharge at a port rather than by time. Although the column generation formulation allows them to implicitly consider all possible routes and load/discharge quantities, the sequence indexed formulation unfortunately requires several assumptions including constant production/demand rates, and constant port capacities. Furthermore, the dynamic program (DP) they use to solve the pricing problem (i.e. generate negative reduced cost routes and assign load/discharge quantities for ports visited on the respective routes), uses a coarse discretization of load/discharge quantities relative to vessel capacity. Although their master problem uses a convex combination of these load/discharge quantities generated from the pricing problem to determine the actual quantities, the artificial discretization loses any guarantee of producing a true bound on the optimal solution. In addition, the discretization is also used to reduce the number of possible loads/discharges at a port, heuristically reducing the size of the

problem even further.

In this chapter we develop a branch-price-and-cut algorithm for IRP that combines the flexibility of a time indexed formulation with column generation. Although developed with maritime transportation in mind, the developments in this chapter address a more general IRP than previously considered in the literature. As in Christiansen's approach [26], the pricing problem generates not only the routes but also, the load and discharge quantities for ports visited on the route. However, since we use a time indexed formulation, we do away with any assumption related to constant capacities and constant production/demand rates. Additionally, we include other practical considerations such as draft limits and demurrage costs. More importantly, we use a DP algorithm for the pricing problem that generates certain "extremal" load/discharge quantities. This allows a convex combination of these quantities within the master problem to be sufficient to guarantee a bound on the value of an optimal solution. Despite using a column generation formulation, the LP bounds produced are relatively weak compared to examples of pure 0-1 column generation. To further tighten the formulation, we use the problem's boundary conditions in preprocessing and to restrict the set of columns that are produced by the pricing problem. We also introduce branching schemes and cuts that can be implemented efficiently and that preserve the structure of the pricing problem. Some of the cuts are inspired by the capacity cuts known for the vehicle routing problem. Although these cuts are useful to tighten the relaxation, they are somewhat limited since they are derived from purely binary implications. We also derive a new class of mixed 0-1 cuts that considers both binary and continuous variables specifically targeting fractional solutions brought about by individual vessels "competing" for limited inventory at load ports and limited storage capacity at discharge ports. In all, the proposed branch-price-and-cut approach contains several innovations including the pricing and the cuts allowing us to solve practically sized problems to optimality and to produce reasonable bounds for harder

instances when alternative branch-and-cut approaches fail.

4.2 A Column Generation Formulation

Given a set of supply and demand facilities denoted by J_S and J_D respectively, IRP is concerned with the distribution of a product from facilities in J_S to facilities in J_D over a planning horizon of length T. Each supply/demand facility $j \in J_S \cup J_D$ produces/consumes $b_{j,t}$ units and has storage capacity for up to a maximum of $Q_{j,t}$ units of inventory during time period $t \in \{1, \ldots, T\}$. The amount of inventory available at the start of the planning horizon is given by $I_{j,0}$. At any $j \in J_S \cup J_D$, the minimum and maximum amount of product that can be loaded/unloaded during a single time period is given by $F_j^{\min}$ and $F_j^{\max}$ respectively. A set of heterogeneous vessels V is available for the distribution of the product. Each vessel $v \in V$ has capacity given by Q_v.

To model IRP, we use a time indexed formulation in which we implicitly consider all possible routes and load/discharge quantities using column generation. We define R_v to be the set of all possible routes for vessel v including the load/discharge quantities at the ports visited on individual routes. For a given route $r \in R_v$, we define $z_{j,t}^{v,r}$ to be the 0-1 indicator corresponding to the load/discharge decision at port j and time t, $f_{j,t}^{v,r}$ ($f_{j,t}^{v,r} \geq 0$ if $j \in J_S$ and $f_{j,t}^{v,r} \leq 0$ if $j \in J_D$) to be the amount of product loaded/discharged at port j and time t, and $c^{v,r}$ to be the cost of the route including any transportation costs, demurrage costs, and expense and/or revenue from procuring/supplying the product. The column generation master problem can then

be stated as

$$\min \sum_{v \in V} \sum_{r \in R_v} c^{v,r} \lambda^{v,r}$$

$$\text{s.t. } I_{j,t} = I_{j,t-1} + b_{j,t} - \sum_{v \in V} \sum_{r \in R_v} f_{j,t}^{v,r} \lambda^{v,r}, \quad j \in J_S, t = 1, \ldots, T, \tag{4.1}$$

$$I_{j,t} = I_{j,t-1} - b_{j,t} - \sum_{v \in V} \sum_{r \in R_v} f_{j,t}^{v,r} \lambda^{v,r}, \quad j \in J_D, t = 1, \ldots, T, \tag{4.2}$$

$$0 \leq I_{j,t} \leq Q_{j,t}, \qquad j \in J_S \cup J_D, t = 1, \ldots, T, \tag{4.3}$$

$$\sum_{r \in R_v} \lambda^{v,r} = 1, \qquad v \in V, \tag{4.4}$$

$$\lambda^{v,r} \geq 0, \qquad v \in V, r \in R_v, \tag{4.5}$$

$$\sum_{r \in R_v} z_{j,t}^{v,r} \lambda^{v,r} \in \{0, 1\}, \qquad v \in V, j \in J_S \cup J_D, t = 1, \ldots, T. \tag{4.6}$$

Constraints (4.1) and (4.2) are the lot-sizing constraints for the supply and demand ports respectively while constraints (4.3) ensure that inventory at a port never exceeds storage capacity or is never depleted. Note that safety stock can be incorporated by imposing a nonzero lower bound on the inventory variables in (4.3). Constraints (4.4) and (4.5) allows a convex combination of routes and load/discharge quantities to be chosen for each vessel, while constraint (4.6) ensures that the chosen convex combination is integer in terms of decisions to load/discharge at a port.

If two routes r_1 and r_2 for vessel v have different load/discharge patterns (i.e., $z_{j,t}^{v,r_1} \neq z_{j,t}^{v,r_2}$ for some j and t), then any solution in which λ^{v,r_1} and λ^{v,r_2} are both positive violates the integrality constraint (4.6) (i.e. will result in $0 < \sum_{r \in R_v} z_{j,t}^{v,r} \lambda^{v,r} < 1$). If we ignore routes that visit a port but do not load/discharge at it, then routes with the same load/discharge pattern may be different only in the load/discharge quantities and in the timing (but not duration) of any demurrage time. Thus, if the expense incurred or revenue obtained from procuring/supplying product to a particular supply/demand location and time point is linear in the amount of product procured/supplied, and demurrage costs are linear with respect to demurrage time,

then enforcing integrality on the load/discharge decisions is sufficient to recover a solution for the original problem.

Given dual values $\pi_{j,t}$ for constraints (4.1) and (4.2), and α_v for constraints (4.4), obtained from solving the linear relaxation of the master problem (i.e. ignoring integrality constraints (4.6)), the column generation pricing problem for vessel v involves finding a route and load/discharge quantities with minimum reduced cost, i.e. for a given vessel v finding

$$\arg\min_{r \in R_v} \left\{ c^{v,r} - \left(\alpha_v + \sum_{j \in J_S \cup J_D} \sum_{t=1}^{T} \pi_{j,t} f_{j,t}^{v,r} \right) \right\}.$$

Since we do not explicitly keep on hand all possible routes and associated load/discharge quantities, we need to be able to dynamically construct routes and assign load/discharge quantities so that the resulting column has minimum reduced cost. To do this, we use a time expanded network $\mathcal{N}_v = (N_v, A_v)$ (N_v is the set of nodes and A_v is the set of arcs) in which a path from some source to sink maps the route of vessel v over the entire planning horizon. Each node $n = (j,t) \in N_v$ corresponds to a location $j \in J_S \cup J_D$ and time point $t \in \{1, \ldots, T\}$. An arc $e = (n_1 = (j_1, t_1), n_2 = (j_2, t_2)) \in A_v$ corresponds to either the relocation of a vessel from one port to another (i.e. $j_1 \neq j_2$ and $t_2 - t_1 > 0$ is the time taken to travel between j_1 and j_2), or a vessel remaining demurraged at a port (i.e. $j_1 = j_2$ and $t_2 = t_1 + 1$). The fixed transportation and demurrage costs are included in the cost c_e of an arc. Additionally, for each node $n = (j,t) \in N_v$, any expense incurred or revenue obtained from procuring/supplying a single unit of product at port j and time t including the dual values $\pi_{j,t}$ are included in the per unit cost $\bar{c}_n$ on the amount loaded/discharged at port j and time t. For each node $n = (j,t) \in N_v$, we define $U_n \leq Q_v$ to be the maximum amount of inventory a vessel can have on board when entering port j at time t. In addition to ensuring vessel capacity is never exceeded, U_n can also be used to incorporate additional practical constraints such as draft limits that vary by port, and requiring a vessel to have no

inventory left over at the end of its route. For each node $n = (j, t) \in N_v$, we also define $F_n^{\min} = F_j^{\min}$ and $F_n^{\max} = F_j^{\max}$ if $j \in J_S$, and $F_n^{\min} = -F_j^{\max}$ and $F_n^{\max} = -F_j^{\min}$ if $j \in J_D$ to be the bounds on the load/discharge quantity at port j and time t. Given source and sink nodes n_s and n_t respectively corresponding to the earliest and latest time points a vessel is available for use, the pricing problem for vessel v can then be stated as

$$\min \sum_{e \in A_v} c_e x_e + \sum_{n \in N_v} \bar{c}_n f_n$$

$$\text{s.t.} \sum_{e=(n,n') \in A_v} x_e - \sum_{e=(n',n) \in A_v} x_e = \begin{cases} +1 & \text{if } n = n_s, \\ -1 & \text{if } n = n_t, \\ 0 & \text{if } n \in N_v \setminus \{n_s, n_t\}, \end{cases} \tag{4.7}$$

$$z_n \leq \sum_{e=(n,n') \in A_v} x_e \qquad n \in N_v, \tag{4.8}$$

$$F_n^{\min} z_n \leq f_n \leq F_n^{\max} z_n \qquad n \in N_v, \tag{4.9}$$

$$F_n = x_e \left(F_{n'} + f_{n'}\right) \qquad e = (n', n) \in A_v, \tag{4.10}$$

$$0 \leq F_n \leq U_n \qquad n \in N_v, \tag{4.11}$$

$$x_e \in \{0, 1\} \qquad e \in A_v, \tag{4.12}$$

$$z_n \in \{0, 1\} \qquad n \in N_v. \tag{4.13}$$

Constraints (4.7) ensure that x_e induces a path in $\mathcal{N}_v$ from source to sink while constraints (4.8) ensure that for each node $n \in N_v$, the decision z_n to load/discharge at the port and time associated with n is at value one only if n is visited along the path. Constraints (4.9) ensures that f_n, the amount loaded/discharged at n, is between the required bounds while constraints (4.10) and (4.11) ensure that F_n, the amount of inventory on board the vessel when entering the port associated with n, is between 0 and U_n. Note that constraints (4.10) can be linearized using standard "big M" techniques (see Glover [69]).

In general, the above pricing problem involves finding a shortest path in a network

from source to sink in which the amount of product loaded/discharged when visiting individual nodes along the path is a continuous bounded variable with linear cost. Note that if $F_n^{\min} = F_n^{\max}$ (i.e. the amount of product loaded/discharged when visiting a node is fixed) for all n, then the pricing problem resembles a more traditional *resource constrained shortest path problem* (RCSPP) (see Irnich and Desaulniers [77]). Otherwise, if $F_n^{\min} < F_n^{\max}$ for one or more nodes n, the resulting pricing problem is a variant of the *fixed charge shortest path problem* (FCSPP) studied in Chapter 3 in which resource consumption is semicontinuous and we have both lower and upper bounds on the total amount of resource that is accumulated up to each node that is visited.

As a final note, we point out that if the columns (i.e. routes and associated load/discharge quantities) added to the problem correspond to extreme points of the convex hull of feasible solutions to the above pricing problem, then the master problem can be seen as being obtained as a result of performing Dantzig-Wolfe decomposition ([37]) on the corresponding compact arc flow formulation. The compact arc flow formulation, analogous to the formulation of Savelsbergh and Song [99] for IRP with continuous moves, can be obtained by replacing the columns corresponding to the routes and load/discharge quantities in the master problem with explicit routing and load/discharge decisions as in the above formulation for the pricing problem for each individual vessel. As is typical when the original compact formulation has mixed integer variables, integrality in the corresponding column generation formulation is often required on auxiliary binary variables rather than (as is more common in pure 0-1 column generation) on the column variables. We refer the reader to Vanderbeck [112], and Poggi De Aragão and Uchoa [89] for the general decomposition principle for branch-price-and-cut, and Desaulniers [39] and Christiansen [26] for examples detailing the steps of the decomposition scheme and the resulting integrality requirements for mixed integer compact formulations for the split delivery vehicle routing problem

and a sequence indexed formulation for IRP respectively.

4.3 Solution Method

We use a branch-price-and-cut algorithm to solve the master problem. Within this framework, we solve a linear relaxation of the master problem at each node in the branch-and-bound tree through column generation. If the solution at a node is fractional (i.e. does not satisfy conditions (4.6)), we attempt to add cuts to eliminate the fractional solution and/or branch by partitioning the feasible space of integer solutions in a way that eliminates the fractional solution. Both the cutting planes and branching decisions are made in the space of the compact arc flow formulation, thus preserving the structure of the pricing problem.

4.3.1 Solving the Pricing Problem

Given network $\mathcal{N}_v = (N_v, A_v)$ with associated bounds U_n, $F_n^{\min}$, and $F_n^{\max}$ described earlier for each node $n \in N_v$, the pricing problem is one of finding a minimum cost path from source to sink and associated load/discharge quantities for ports visited along the path. We assume that the appropriate dual values are embedded in the cost c_e of arcs $e \in A_v$ and within the per unit cost $\bar{c}_n$ of loading/discharging product at the port and time point corresponding to node $n \in N_v$ and thus, we do not make a distinction between minimum cost and minimum reduced cost solutions to the pricing problem. In the remaining part of this section we outline a DP algorithm for the pricing problem. We start by identifying certain properties integral to the design of the algorithm.

For a given path P where $N(P) = \{n_0, n_1, n_2, \ldots, n_K\}$ is the sequence of nodes visited by P, and given load/discharge decisions $z_{n_i} \in \{0, 1\}$ for $i = 0, \ldots, K$, the

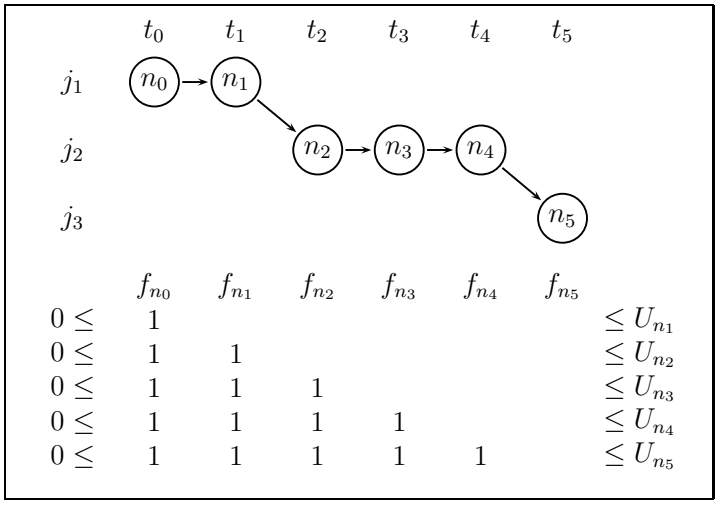


Figure 4.1: The structure of the knapsack constraints limiting the total amount of inventory on the vessel along path P.

optimal load/discharge quantities along P can be obtained by solving the LP

$$\min \sum_{i=0,\ldots,K} \bar{c}_{n_i} f_{n_i}$$

$$\text{s.t. } 0 \leq \sum_{i=0,\ldots,k-1} f_{n_i} \leq U_{n_k}, \quad k = 1, \ldots, K, \tag{4.14}$$

$$F^{\min}_{n_i} z_{n_i} \leq f_{n_i} \leq F^{\max}_{n_i} z_{n_i},\ i = 0, \ldots, K. \tag{4.15}$$

This LP admits a nested structure (see Figure 4.1) similar to the linear relaxation of the *multi-period knapsack problem* described in Faaland [55] and Dudzinski and Walukiewicz [49]. The following proposition describes the optimal load/discharge quantities along P.

Proposition 4.3.1. *If there exists a feasible assignment of load/discharge quantities along P, then there exists an optimal assignment $f^*_{n_k}$ $k = 0, \ldots, K$ such that for each $k = 0, \ldots, K$ either*

i. $f^*_{n_k} \in \{0, F^{\min}_{n_k}, F^{\max}_{n_k}\}$,

ii. $f^*_{n_k} = U_{n_i} - \sum_{j \in \{0,\ldots,i-1\} \setminus \{k\}} f^*_{n_j}$ *for some* $i > k$ *and* $f^*_{n_j} \in \{0, F^{\min}_{n_j}, F^{\max}_{n_j}\}$ *for all* $j \in \{k+1, \ldots, i-1\}$, *or*

iii. $f^*_{n_k} = -\sum_{j \in \{0,\ldots,i-1\} \setminus \{k\}} f^*_{n_j}$ *for some* $i > k$ *and* $f^*_{n_j} \in \{0, F^{\min}_{n_j}, F^{\max}_{n_j}\}$ *for all* $j \in \{k+1, \ldots, i-1\}$.

Proof. Proposition 4.3.1 simply states that there exists optimal load/discharge quantities such that each node n_k with load/discharge quantity strictly between its lower and upper bounds (i.e. such that $F^{\min}_{n_k} < f^*_{n_k} < F^{\max}_{n_k}$) is followed by a node n_i ($i > k$) such that the total amount loaded/discharged before reaching i is either 0 or U_{n_i}, and the load/discharge quantities for nodes between k and i is either 0, at their lower bound, or at their upper bound, i.e. $f^*_{n_j} \in \{0, F^{\min}_{n_j}, F^{\max}_{n_j}\}$ for all $j = k+1, \ldots, i-1$. Indeed, if the above conditions do not hold for some optimal allocation of load/discharge quantities along P, then it must be the case that there either exists k_1 such that

i. $F^{\min}_{n_{k_1}} < f^*_{n_{k_1}} < F^{\max}_{n_{k_1}}$, and

ii. $0 < \sum_{j=0,\ldots,i-1} f^*_{n_j} < U_{n_i}$ for all $i = k_1 + 1, \ldots, K$,

or there exists k_1 and k_2 ($k_1 < k_2$) such that

i. $F^{\min}_{n_{k_1}} < f^*_{n_{k_1}} < F^{\max}_{n_{k_1}}$,

ii. $F^{\min}_{n_{k_2}} < f^*_{n_{k_2}} < F^{\max}_{n_{k_2}}$, and

iii. $0 < \sum_{j=0,\ldots,i-1} f^*_{n_j} < U_{n_i}$ for all $i = k_1 + 1, \ldots, k_2 + 1$.

Note that for the second scenario and $\varepsilon > 0$ small enough, we remain feasible by increasing $f^*_{n_{k_1}}$ by ε and decreasing $f^*_{n_{k_2}}$ by ε. Thus, we must have $\bar{c}_{n_{k_1}} = \bar{c}_{n_{k_2}}$. Furthermore, we can gradually increase ε until we either reach the upper limit for the quantity loaded/discharged at n_{k_1}, the lower limit for the quantity loaded/discharged at n_{k_2}, or the knapsack bound of 0 or U_{n_i} for the inventory on the vessel before entering node n_i for some $i = k_1 + 1, \ldots, k_2 + 1$. Analogous arguments can be used for the first scenario and thus, we may assume that there always exists an optimal allocation of load/discharge quantities as given by Proposition 4.3.1. □

Using Proposition 4.3.1, we are able to construct an optimal solution using dynamic programming over a pseudo-polynomial state space. Consider a path P from source to sink with $N(P) = \{n_0, n_1, \ldots, n_K\}$, and load/discharge quantities $f_{n_0}, f_{n_1}, \ldots, f_{n_K}$ satisfying conditions of Proposition 4.3.1. We show that one can obtain a label corresponding to a path from source to sink that dominates the label corresponding to P and associated load/discharge quantities by only considering non-dominated labels from various recursions of standard RCSPP in a slightly modified network and appropriate concatenations of these labels.

Let $\vec{\mathcal{N}}_v = (N_v, \vec{A}_v)$ be the network obtained from $\mathcal{N}_v$ by replacing each arc $e = (n_1, n_2) \in A_v$ with three parallel arcs corresponding to three possibilities for the amount loaded/discharged at n_1:

i. the first corresponds to a decision not to load/discharge at n_1,

ii. the second corresponds to loading/discharging exactly $F^{\min}_{n_1}$ at n_1, and

iii. the third corresponds to loading/discharging exactly $F^{\max}_{n_1}$ at n_1.

If $f_{n_j} \in \{0, F^{\min}_{n_j}, F^{\max}_{n_j}\}$ for all $j = 0, \ldots, K$, then the solution can be obtained by using standard labeling techniques for RCSPP (see for example Irnich and Desaulniers [77]) in $\vec{\mathcal{N}}$. Otherwise, let n_{k_1} be the first node of P such that $F^{\min}_{n_{k_1}} < f_{n_{k_1}} < F^{\max}_{n_{k_1}}$. From Proposition 4.3.1 it follows that there exists $i_1 > k_1$ such that $\sum_{j=0,\ldots,i_1-1} f_{n_j} = T_{n_{i_1}} \in \{0, U_{n_{i_1}}\}$ and $f_{n_j} \in \{0, F^{\min}_{n_j}, F^{\max}_{n_j}\}$ for all $j = k_1 + 1, \ldots, i_1 - 1$. Furthermore, since n_{k_1} is also the first node in P where the load/discharge quantity is strictly between its lower and upper limits, we must also have $f_{n_j} \in \{0, F^{\min}_{n_j}, F^{\max}_{n_j}\}$ for all $j = 0, \ldots, k_1 - 1$. Let $f_{L_1} = \sum_{j=0,\ldots,k_1-1} f_{n_j}$ be the amount of inventory on the vessel before entering n_{k_1} and $f_{R_1} = T_{n_{i_1}} - \sum_{j=k_1+1,\ldots,i_1-1} f_{n_j}$ be the amount of inventory on the vessel before entering n_{k_1+1}. Furthermore, let c_{L_1} be the minimum cost of a path from n_0 to n_{k_1} in $\vec{\mathcal{N}}$ that has f_{L_1} units of inventory before entering n_{k_1}. Similarly, let c_{R_1} be the minimum cost of a path from n_{k_1+1} to n_{i_1} in $\vec{\mathcal{N}}$ that

has f_{R_1} units of inventory before entering n_{k_1+1} and $T_{n_{i_1}}$ units of inventory before entering node n_{i_1}. Note that the label L_1 corresponding to the path from n_0 to n_{k_1} and associated load/discharge quantities, resulting in f_{L_1} units of inventory before entering n_{k_1} and cost c_{L_1}, can be obtained using standard labeling techniques in $\vec{\mathcal{N}}_v$ when starting with the trivial path rooted at the source and 0 inventory. Similarly, the label R_1 corresponding to the path from n_{k_1+1} to n_{i_1} and associated load/discharge quantities, resulting in f_{R_1} units of inventory before entering n_{k_1}, T_{n_i} units before entering n_{i_1}, and cost c_{R_1}, can be obtained using standard labeling techniques in $\vec{\mathcal{N}}_v$ when starting with the trivial path rooted at n_{i_1}, an inventory of T_{n_i}, and working our way backwards towards the sink (i.e. extending labels with incoming arcs rather than outgoing arcs). Thus, a path from n_0 to n_{i_1} and associated load/discharge quantities that has $T_{n_{i_1}}$ units of inventory on the vessel before entering n_{i_1} (i.e. the same amount of inventory as the subpath of P from n_0 to n_{i_1} and its associated load/discharge quantities) with at worst the same cost as the subpath of P from n_0 to n_{i_1} and its associated load/discharge quantities can be obtained by concatenating L_1 and R_1. The load/discharge quantity at n_{k_1} as a result of the concatenation is $f_{R_1} - f_{L_1}$, and the cost of the resulting label L_2 is $c_{L_1} + c_{R_1} + c_e + \bar{c}_{n_{k_1}}(f_{R_1} - f_{L_1})$ where $e = (n_{k_1}, n_{k_1+1}) \in A_v$ (see Figure 4.2). Note that $f_{n_k} = f_{R_1} - f_{L_1}$ and thus, $F^{\min}_{n_{k_1}} < f_{R_1} - f_{L_1} < F^{\max}_{n_{k_1}}$.

Given L_2 corresponding to a path from n_0 to n_{i_1} that dominates the label corresponding to the sub-path of P from n_0 to n_{i_1} and associated load/discharge quantities before entering n_{i_1}, we then examine the next node n_{k_2} ($k_2 \geq i_1$) for which $F^{\min}_{n_{k_2}} < f_{n_{k_2}} < F^{\max}_{e_{k_2}}$. Since k_2 is the first node in P after k_1 where resource consumption is strictly between its lower and upper limits, we must have $f_{n_j} \in \{0, F^{\min}_{n_j}, F^{\max}_{n_j}\}$ for all $j = i_1, \ldots, k_2 - 1$. Thus, the label corresponding to the sub-path of P from n_0 to n_{k_2} and associated load/discharge quantities before entering n_{k_2} is dominated by a label say L_3 obtained by extending L_2 along nodes $n_{i_1+1}, \ldots, n_{k_2-1}$, and with

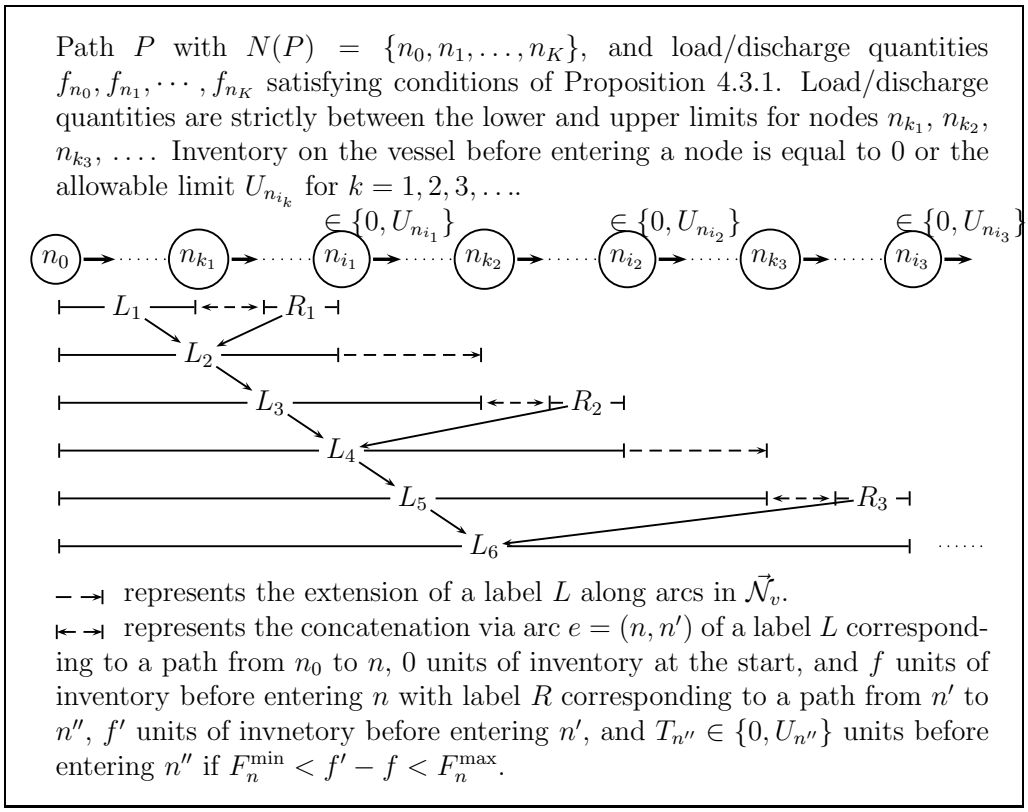


Figure 4.2: Constructing a path whose label dominates the label associated with a given path from source to sink and load/discharge quantities satisfying Proposition 4.3.1.

load/discharge quantities at either 0, or their lower or upper limits, i.e. extending L_2 along arcs in $\vec{\mathcal{N}}_v$. Furthermore, from Proposition 4.3.1 we again have that there exists $i_2 > k_2$ such that $\sum_{j=1,\ldots,i_2-1} f_{n_j} = T_{n_{i_2}} \in \{0, U_{n_{i_2}}\}$ and $f_j \in \{0, F^{\min}_{n_j}, F^{\max}_{n_j}\}$ for all $j = k_2 + 1, \ldots, i_2 - 1$. Hence, the label corresponding to the sub-path of P from n_0 to n_{i_2} and load/discharge quantities before entering n_{i_2} is dominated by some label say L_4 obtained by concatenating L_3 with some non-dominated label say R_2 corresponding to a path from n_{k_2+1} to n_{i_2} generated during the backward recursion in $\vec{\mathcal{N}}$ rooted at n_{i_2} and starting with inventory $T_{n_{i_2}}$.

As shown in Figure 4.2, continuing in the above fashion for each node of P with load/discharge quantities strictly between their respective lower and upper limits, we can construct a path and assign load/discharge quantities so that the corresponding label dominates the label corresponding to P and its associated load/discharge quantities. Thus, the pricing problem can be solved by solving $2|N_v|$ backward recursions of standard RCSPP in $\vec{\mathcal{N}}_v$, two for each node $n \in N_v$, one starting with a trivial path at n and inventory of 0, and the other starting with a trivial path at n and an inventory of U_n, and by further combining the nondominated labels generated during these recursions within a single forward recursion of RCSPP in $\vec{\mathcal{N}}$ rooted at the source. We refer the reader to Chapter 3 for further details about the proposed DP algorithm including additional schemes to improve its efficiency, its merits relative to alternative approaches and an extensive computational study including experiments on the pricing problem for the inventory routing problem, the split delivery vehicle routing problem, and a machine scheduling problem with controllable processing times.

As with most column generation implementations, we use a heuristic version of the DP algorithm to construct negative reduced cost columns quickly and revert to the exact algorithm only when these heuristics fail. Since we have varying bounds on the amount of inventory allowed on the vessel along the path that maps its route, including requiring no inventory to be left over at the end of the route, solving the

pricing problem using standard RCSPP techniques with an artificial discretization of load/discharge quantities does not provide good solutions unless the discretization is relatively fine. Instead of discretizing the load/discharge quantities, we restrict the length (in number of arcs) of paths generated during the backward recursion in $\vec{\mathcal{N}}_v$, i.e. we do not extend a backward label if it corresponds to a path of a certain length. Hence, if we restrict the length of paths generated during the backward recursion to k, we essentially give a forward label $k+1$ load/discharge opportunities to adjust for any upcoming bound on the amount of inventory allowed on the vessel. In terms of the trade-off between quality of solution and run time, we find that restricting paths to a length of at most 2 arcs during the backward recursions works best as a heuristic.

Note that the load/discharge quantities generated during the pricing problem do not necessarily correspond to feasible load/discharge quantities that a vessel in an integer solution may adopt since the pricing problem does not consider the actual available inventory at load ports and remaining storage capacity at discharge ports on account of activities of other vessels. Of course, this level of coordination is considered within the master problem which ensures that the convex combinations used lead to feasible load/discharge quantities for each vessel and is also globally feasible over all vessels, production/demand rates and port capacities. However, we can restrict the set of feasible columns by preprocessing and adding additional constraints within the pricing problem by exploiting storage capacities, production/demand rates, and boundary conditions for inventory to tighten the "extremal" load/discharge quantities.

The bounds on the maximum amount that can be loaded/discharged at node $n = (j, t)$ can be tightened beyond $F_j^{\max}$. Indeed, for $j \in J_S$, $F_n^{\max}$ can be tightened to the minimum of $F_j^{\max}$, Q_v, $Q_{j,t-1} + b_{j,t}$, and

$$I_{j,0} + \sum_{t'=1,\ldots,t} b_{j,t'} - \left\lceil \frac{\max\left\{0, I_{j,0} + \sum_{t'=1,\ldots,t-1} b_{j,t'} - Q_{j,t}\right\}}{F_j^{\max}} \right\rceil F_j^{\min} \tag{4.16}$$

i.e. the maximum amount of inventory available at port j and time t with only the necessary number of loads before t to ensure storage capacity at j is not exceeded. Similarly, for $j \in J_D$, $F_n^{\min}$ can be tightened to the maximum of: $-F_j^{\max}$, $-Q_v$, $-Q_{j,t-1} - b_{j,t}$, and

$$-\left(Q_{j,t} - I_{j,0} + \sum_{t'=1,\ldots,t} b_{j,t'} + \left\lceil \frac{\max\left\{0, -I_{j,0} + \sum_{t'=1,\ldots,t-1} b_{j,t'}\right\}}{F_j^{\max}} \right\rceil F_j^{\min}\right) \tag{4.17}$$

i.e. the maximum amount of vacant storage capacity available at port j and time t with only the necessary number of discharges before t to ensure inventory is never depleted.

This preprocessing only tightens the amount loaded/discharged at a port for a given time point. The quantity

$$I_{j,0} + \sum_{t'=1,\ldots,t-1} b_{j,t'} \tag{4.18}$$

however, is a bound on the total amount of product that can be loaded at port $j \in J_S$ before time t. Similarly, the quantity

$$Q_{j,t} - I_{j,0} + \sum_{t'=1,\ldots,t-1} b_{j,t'} \tag{4.19}$$

is a bound on the total amount of product that can be discharged at port $j \in J_D$ before time t. These bounds cannot be enforced by simply tightening the bound U_n for node $n = (j,t)$ since U_n is the amount of inventory on the vessel before entering port j and time t, i.e. it is the net amount of product loaded/discharged over all ports visited. Unfortunately, adding additional constraints restricting the amount loaded/discharged at each port that is visited disrupts the nested structure of the multi-period knapsack problem (see Figure 4.3) leaving us without the properties established in Proposition 4.3.1. However, note that these additional constraints are indeed nested within the original structure if we only consider the first port that is visited along the path. Thus, enforcing (4.16) at node n corresponding to the first

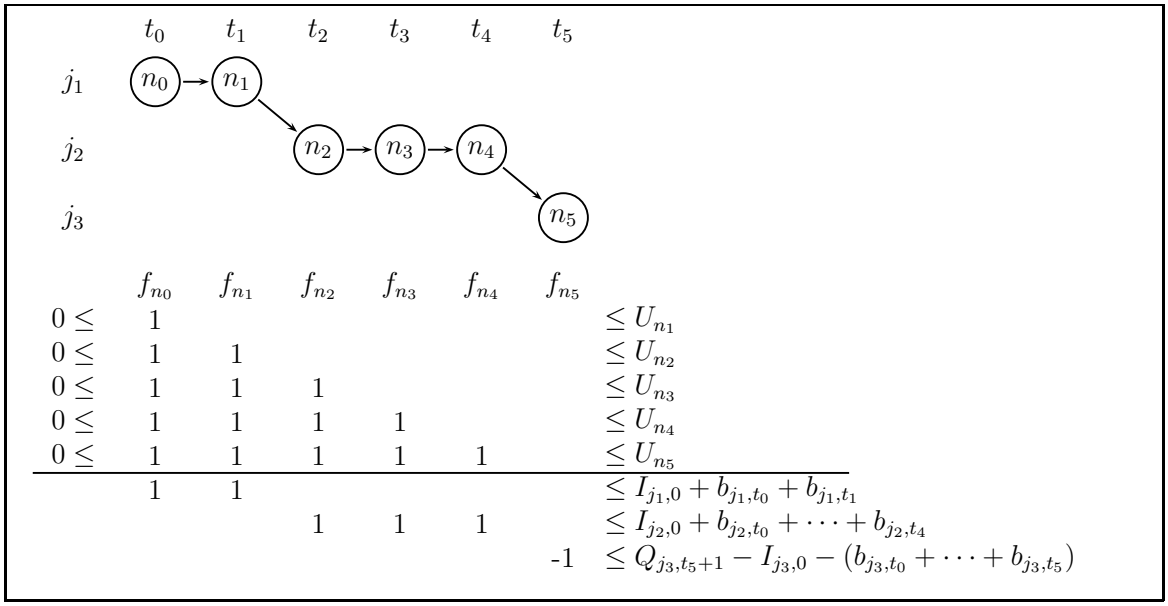


Figure 4.3: The structure of constraints limiting the total amount of inventory on the vessel and the amount loaded/discharged at each port $j_1, j_2 \in J_S$ and $j_3 \in J_D$ visited along path P.

port that is visited maintains the property that a node with load/discharge quantity strictly between its allowable limits is followed by nodes with load/discharge quantities at 0, the lower limit, or the upper limit, and ultimately by some node where one of the corresponding knapsack constraints holds tight.

To restrict the amount of inventory loaded at the first port that is visited, we initiate a backward recursion starting with three (rather than the usual two) initial inventories including 0, U_n, and the bound given by (4.16) for each node n corresponding to a load port. When extending the label backwards or when concatenating, we simply ensure that labels corresponding to a backward recursion rooted at n and start inventory of 0 or U_n have at least one relocation before reaching n while for labels corresponding to a backward recursion rooted at n and start inventory given by (4.16), we ensure there are no relocations before n. As time progresses, the bounds given by equations (4.16) and (4.17) for the total amount loaded/discharged becomes weaker and ultimately redundant (i.e. dominated by U_n). Thus, even if we could

impose these additional constraints on the amount loaded/discharged at every port that is visited, this would further eliminate only a small number of columns that may be otherwise included in the master problem.

As a final note on the pricing problem, we note that we force a vessel to load/discharge immediately before relocating. Not only does this speed up the pricing process as we consider fewer options for load/discharge patterns, but more importantly, this alleviates some of the symmetry between columns corresponding to the same load/discharge pattern but with different demurrage times.

4.3.2 Generating Cutting Planes

To tighten the linear relaxation of the master problem, we consider two families of cuts. The first of these uses vessel and storage capacity, and production/demand rates to bound the number of loads/discharges or visits to a port or set of ports not unlike the intention of the capacity cuts used for the vehicle routing problem (see Lysgaard et al. [84]). The second class, a mixed 0-1 cut, uses vessel capacity and production/demand rates to determine the timing of loads/discharges and/or visits to a port.

We introduce the following notation to make it easier to state these cuts.

- $Q = \max_{v \in V} Q_v$ is the maximum capacity of any vessel,
- $Q_j(t_1, t_2) = \max_{t=t_1,\ldots,t_2} Q_{j,t}$ is the maximum storage capacity at port j during time interval $[t_1, t_2]$,
- $b_j(t_1, t_2) = \sum_{t=t_1,\ldots,t_2} b_{j,t}$ is the total amount of product produced/consumed at port j during the time interval $[t_1, t_2]$,
- given a set of ports $J \subseteq J_S \cup J_D$, and time interval $[t_1^j, t_2^j]$ for each $j \in J$, $N_v(J, (t_1^j, t_2^j)_{j \in J})$ is the set of all nodes $n = (j, t) \in N_v$ such that $j \in J$ and $t_1^j \leq t \leq t_2^j$,

- for some subset of nodes $\bar{N} \subseteq N_v$, $A_v^+(\bar{N})$ is the set of arcs with tail in $\bar{N}$ and head not in $\bar{N}$,
- $z_j(t_1, t_2) = \sum_{v \in V} \sum_{r \in R_v} \sum_{t=t_1,\ldots,t_2} z_{j,t}^{v,r} \lambda^{v,r}$ is the number of loads/discharges at port j during the time interval $[t_1, t_2]$,
- $x_e = \sum_{r \in R_v} x_e^{v,r} \lambda^{v,r}$ where $x_e^{v,r}$ is a 0-1 indicator corresponding to whether route r of vessel v corresponds to a path in $\mathcal{N}_v$ that uses arc e, and
- $f_j(t_1, t_2) = \sum_{v \in V} \sum_{r \in R_v} \sum_{t=t_1,\ldots,t_2} f_{j,t}^{v,r} \lambda^{v,r}$ is the amount of product loaded/discharged at port j during the time interval $[t_1, t_2]$.

Note that the dual value corresponding to any cut containing nonzero coefficients associated with x_e can be embedded in the fixed cost c_e of e. Similarly, the dual value corresponding to any cut containing nonzero coefficients associated with $z_{j,t}^{v,r}$ can be embedded in the fixed cost of arcs $e = (n_1, n_2) \in \vec{\mathcal{N}}_v$ corresponding to loading at least $F_j^{\min}$ at the tail node $n_1 = (j, t)$. Finally, the dual value corresponding to any cut containing nonzero coefficients associated with $f_{j,t}^{v,r}$ can be embedded in the per unit cost $\bar{c}_n$ of procuring/supplying product at node $n = (j, t)$.

4.3.2.1 Port capacity cuts

For a given load port $j \in J_S$, and time interval $[t_1, t_2]$,

$$\left\lceil \frac{I_{j,t_1-1} + b_j(t_1, t_2) - Q_{j,t_2}}{F_j^{\max}} \right\rceil \tag{4.20}$$

is a lower bound on $z_j(t_1, t_2)$. Indeed, $I_{j,t_1-1} + b_j(t_1, t_2) - Q_{j,t_2}$ is the amount of inventory in excess of storage capacity that needs to be shipped during $[t_1, t_2]$. For $t_1 = 1$, the resulting cut is simply the load cover introduced in Savelsbergh and Song [99]. For $t_1 > 1$, (4.20) leads to a nonlinear bound on $z_j(t_1, t_2)$ since I_{j,t_1-1} is a variable and thus, cannot directly be imposed as a lower bound on $z_j(t_1, t_2)$. Note that the lower bound given by (4.20) without the ceiling operator is implied by the

linear relaxation of the master problem. However, by bounding I_{j,t_1-1}, we can derive two linear cuts that bear a resemblance to Gomory's mixed integer cuts (see Gomory [71]) and that lead to a convex piecewise linear approximation of the step function defined by (4.20) (see Figure 4.4). Let $E_j^{\min}(t_1,t_2) = b_j(t_1,t_2) - Q_{j,t_2}$ be the amount of inventory in excess of storage capacity that needs to be shipped if the inventory at $t_1 - 1$ is 0. Then since $0 \leq I_{j,t_1-1}$,

$$z_j(t_1,t_2) \geq \left\lceil \frac{E_j^{\min}(t_1,t_2)}{F_j^{\max}} \right\rceil \tag{4.21}$$

is a valid inequality for $t_1 > 1$. Similarly, let $E_j^{\max}(t_1,t_2) = Q_{j,t_1-1} + b_j(t_1,t_2) - Q_{j,t_2}$ be the amount of inventory in excess of storage capacity that needs to be shipped if the inventory at $t_1 - 1$ is Q_{j,t_1-1}, and

$$\hat{E}_j^{\max}(t_1,t_2) = \left\lfloor \frac{E_j^{\max}(t_1,t_2)}{F_j^{\max}} \right\rfloor F_j^{\max}.$$

Then since $I_{j,t_1-1} \leq Q_{j,t_1-1}$,

$$z_j(t_1,t_2) \geq \frac{I_{j,t_1-1} + b_j(t_1,t_2) - Q_{j,t_2}}{E_j^{\max}(t_1,t_2) - \hat{E}_j^{\max}(t_1,t_2)} + \left\lceil \frac{E_j^{\max}(t_1,t_2)}{F_j^{\max}} \right\rceil - \frac{E_j^{\max}(t_1,t_2)}{E_j^{\max}(t_1,t_2) - \hat{E}_j^{\max}(t_1,t_2)} \tag{4.22}$$

is a valid inequality for $t_1 > 1$ if $\hat{E}_j^{\max}(t_1,t_2) < E_j^{\max}(t_1,t_2)$ (see Figure 4.4).

Analogous cuts can also be identified for demand ports. For a given demand port $j \in J_D$, and time interval $[t_1,t_2]$,

$$\left\lceil \frac{-I_{j,t_1-1} + b_j(t_1,t_2)}{F_j^{\max}} \right\rceil \tag{4.23}$$

is a lower bound on $z_j(t_1,t_2)$. Indeed, $-I_{j,t_1-1} + b_j(t_1,t_2)$ is the amount of inventory in deficit of the amount demanded during $[t_1,t_2]$. Again, for $t_1 = 1$, the resulting cut is simply the discharge cover introduced in Savelsbergh and Song [99], and for $t_1 > 1$, (4.23) leads to a nonlinear bound on $z_j(t_1,t_2)$. However as before, by bounding I_{j,t_1-1}, we can derive linear cuts corresponding to a convex piecewise linear approximation of the step function given by (4.23). Let $D_j^{\min}(t_1,t_2) = -Q_{j,t_1-1} + b_j(t_1,t_2)$ be the amount

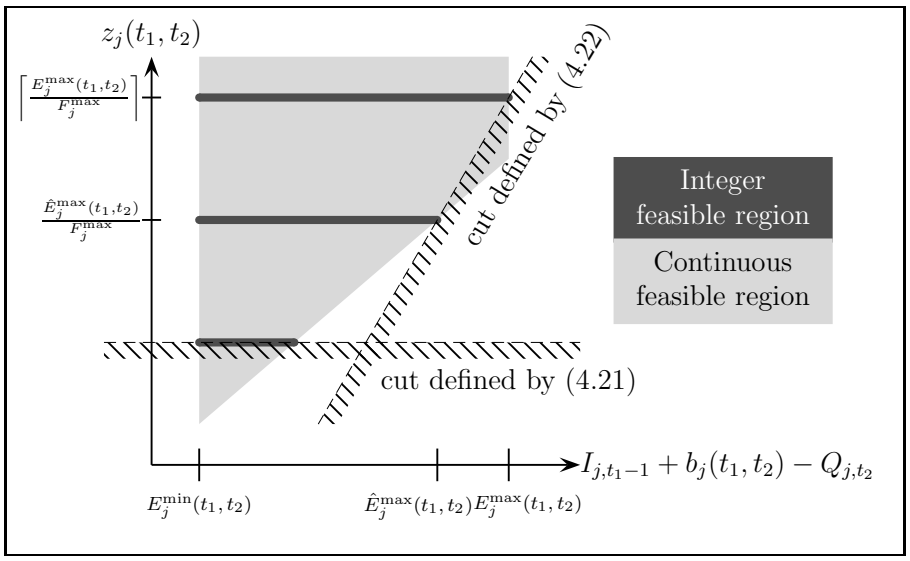


Figure 4.4: Cuts derived from a convex piecewise linear approximation of the lower bound (4.20) on $z_j(t_1, t_2)$.

of inventory in deficit of the amount demanded during $[t_1, t_2]$ when the inventory at $t_1 - 1$ is Q_{j,t_1-1}. Then since $I_{j,t_1-1} \leq Q_{j,t_1-1}$,

$$z_j(t_1, t_2) \geq \left\lceil \frac{D_j^{\min}(t_1, t_2)}{F_j^{\max}} \right\rceil \tag{4.24}$$

is a valid inequality for $t_1 > 1$. Similarly, let $D_j^{\max}(t_1, t_2) = b_j(t_1, t_2)$ be the amount of inventory in deficit of the amount demanded during $[t_1, t_2]$ when the inventory at $t_1 - 1$ is 0, and

$$\hat{D}_j^{\max}(t_1, t_2) = \left\lfloor \frac{D_j^{\max}(t_1, t_2)}{F_j^{\max}} \right\rfloor F_j^{\max}.$$

Then since $0 \leq I_{j,t_1-1}$,

$$z_j(t_1, t_2) \geq \frac{-I_{j,t_1-1} + b_j(t_1, t_2)}{D_j^{\max}(t_1, t_2) - \hat{D}_j^{\max}(t_1, t_2)} + \left\lceil \frac{D_j^{\max}(t_1, t_2)}{F_j^{\max}} \right\rceil - \frac{D_j^{\max}(t_1, t_2)}{D_j^{\max}(t_1, t_2) - \hat{D}_j^{\max}(t_1, t_2)} \tag{4.25}$$

is a valid inequality for $t_1 > 1$ if $\hat{D}_j^{\max}(t_1, t_2) < D_j^{\max}(t_1, t_2)$.

Separating over port capacity cuts is not computationally expensive. For each port and time interval there are at most two possible cuts that can be added. We thus examine all potential ports and time intervals at each node of the branch-and-bound tree until no appreciable number of these cuts is violated.

4.3.2.2 Vessel capacity cuts

The previous set of cuts attempts to eliminate fractional solutions by bounding the number of loads/discharges at a port based on port capacity, production/demand rates, and the maximum that can be loaded/discharged at a port during a single time period. It does not consider vessel capacity and thus, does not consider the number of visits required to load/discharge a certain amount of inventory.

Given a set of load ports $J \subseteq J_S$, and time interval $[t_1^j, t_2^j]$ for each $j \in J$,

$$\sum_{v \in V} \sum_{e \in A_v^+(N_v(J,(t_1^j,t_2^j)_{j \in J}))} x_e \geq \left\lceil \frac{\sum_{j \in J} \left(I_{j,t_1^j - 1} + b_j(t_1^j, t_2^j) - Q_{j,t_2} \right)}{Q} \right\rceil \tag{4.26}$$

is a valid inequality for the number of visits to ports in J during the given time intervals. Like the port capacity cuts, the above inequality is nonlinear when $t_1^j > 1$ for some $j \in J$. However, by bounding the numerator of the right hand side by $\sum_{j \in J} E_j^{\min}$ and $\sum_{j \in J} E_j^{\max}$, we can again find linear cuts corresponding to a convex piecewise approximation of the step function defining the right hand side.

Analogous cuts can also be identified for demand ports. Given a set of demand ports $J \subseteq J_D$, and time intervals $[t_1^j, t_2^j]$ for all $j \in J$,

$$\sum_{v \in V} \sum_{e \in A_v^+(N_v(J,(t_1^j,t_2^j)_{j \in J}))} x_e \geq \left\lceil \frac{\sum_{j \in J} \left(-I_{j,t_1^j - 1} + b_j(t_1^j, t_2^j) \right)}{Q} \right\rceil \tag{4.27}$$

is a valid inequality for the number of visits to ports in J during the given time intervals. By bounding the numerator of the right hand side by $\sum_{j \in J} D_j^{\min}$ and $\sum_{j \in J} D_j^{\max}$, we can find linear cuts corresponding to a convex piecewise linear approximation of the step function defining the right hand side when $t_1^j > 1$ for some $j \in J$.

As for the port capacity cuts, we try to separate vessel capacity cuts for a single port (i.e. when $|J| = 1$) throughout the branch-and-bound tree until we do not find an appreciable number violated. Finding vessel capacity cuts over multiple ports is considerably more expensive because of the number of such possibilities. We find

diminishing returns with respect to the time taken to generate cuts with multiple ports and the resulting improvement of the bound. We thus restrict the search for vessel capacity cuts with multiple ports to the root node only and never consider more than 3 ports at a time.

When separating vessel capacity cuts with multiple ports, we do not need to consider all possible intervals for each port. Indeed, if

$$A_v^+(N_v(J,(t_1^j,t_2^j)_{j\in J})) = \bigcup_{j\in J} A_v^+(N_v(j,(t_1^j,t_2^j))),$$

then the resulting cuts are dominated by the cuts generated from considering each port in J individually. Thus, when selecting ports and time intervals, we use a sequential heuristic in which we pick the start of the time interval for a port based on the start of the time interval chosen for the previous port selected plus the time taken to travel between the two ports. Thus ensuring that for each port $j \in J$, there is at least one arc e such that $e \in A_v^+(N_v(j,(t_1^j,t_2^j)))$ but $e \notin A_v^+(N_v(J,(t_1^j,t_2^j)_{j\in J}))$.

4.3.2.3 Timing cuts

We often encounter fractional solutions in which the time between successive loads/discharges at a port is relatively small given the actual amount loaded/discharged at the port, and the time required for sufficient inventory to buildup or be consumed at the port. These fractional solutions are especially problematic for instances where shipping relatively little inventory over the planning horizon may be feasible since the previous families of cuts are not effective in this case. We next introduce cuts to alleviate some of this fractionality.

Given load port $j \in J_S$ and time interval $[t_1, t_2]$, let

$$b_j^{\max}(t_1,t_2) = \max_{t=t_1,\ldots,t_2} \left\{ \frac{b_j(t_1,t)}{t-t_1+1} \right\}$$

be a constant overestimation of the production rate at j during $[t_1, t_2]$, and suppose that $(k-1)Q < f_j(t_1,t_2) - I_{j,t_1-1} \leq kQ$ for some $k \in \{1,2,3\ldots\}$. Here $f_j(t_1,t_2) -$

I_{j,t_1-1} is the amount of inventory in excess of what is available at t_1-1 that is loaded at j and during $[t_1,t_2]$. Let t^i be the latest time by which more than $(i-1)Q$ units of inventory $(i \leq k)$ is still left to be loaded at j during $[t_1,t_2]$. Note that

$$t^i \geq t_1 - 1 + \left\lceil \frac{f_j(t_1,t_2) - I_{j,t_1-1} - (i-1)Q}{b_j^{\max}(t_1,t_2)} \right\rceil.$$

Thus, since more than $(i-1)Q$ units of inventory is still left to be loaded during $[t^i,t_2]$, and since each vessel has capacity at most Q, at least i visits must be made to port j during $[t^i,t_2]$ for each $i=1,\ldots,k$ resulting in inequality

$$\sum_{v\in V} \sum_{e\in A_v^+(N_v(j,(t^i,t_2)))} x_e \quad \geq \quad i \quad \forall\, i=1,\ldots,k.$$

Since there are no arcs from nodes in $N_v(j,(t^i,t_2))$ to nodes in $N_v(j,(t_1,t_2))\backslash N_v(j,(t^i,t_2))$, it follows that $A^+(N_v(j,(t^i,t_2))) \subseteq A^+(N_v(j,(t_1,t_2)))$. Thus, it follows that there exists k arcs $e_1,\ldots,e_k$ from the set $\cup_{v\in V} A_v^+(N_v(j,(t_1,t_2)))$ such that $x_{e_j}=1$ for all $j=1,\ldots,k$, and $d_{e_j} \geq t^i$ for all $j \geq i$ and $i=1,\ldots,k$ where d_e is the time corresponding to the tail node of arc e. Hence it follows that

$$\begin{aligned} \sum_{j=1,\ldots,k} d_{e_j} \quad &\geq \quad \sum_{i=1,\ldots,k} t^i, \\ \Rightarrow \sum_{j=1,\ldots,k} (d_{e_j} - t_1 + 1) \quad &\geq \quad \sum_{i=1,\ldots,k} (t^i - t_1 + 1). \end{aligned}$$

Since $\{e_1,\ldots,e_k\} \subseteq \cup_{v\in V} A_v^+(N_v(j,(t_1,t_2)))$, and since $d_e \geq t_1$ for all arcs e with tail in $N_v(j,(t_1,t_2))$, it follows that

$$\begin{aligned} \sum_{v\in V} \sum_{e\in A_v^+(N_v(j,(t_1,t_2)))} (d_e - t_1 + 1)x_e \quad &\geq \quad \sum_{i=1,\ldots,k} (t^i - t_1 + 1) \\ &\geq \quad \sum_{i=1,\ldots,k} \left\lceil \frac{f_j(t_1,t_2) - I_{j,t_1-1} - (i-1)Q}{b_j^{\max}(t_1,t_2)} \right\rceil \\ &\geq \quad \sum_{i=1,\ldots,k} \frac{f_j(t_1,t_2) - I_{j,t_1-1} - (i-1)Q}{b_j^{\max}(t_1,t_2)} \\ &= \quad \frac{k(f_j(t_1,t_2) - I_{j,t_1-1})}{b_j^{\max}(t_1,t_2)} + \frac{k(1-k)Q}{2b_j^{\max}(t_1,t_2)}. \end{aligned}$$

Thus,

$$\sum_{v \in V} \sum_{e \in A_v^+(N_v(j,(t_1,t_2)))} (d_e - t_1 + 1) x_e \geq \frac{k\left(f_j(t_1,t_2) - I_{j,t_1-1}\right)}{b_j^{\max}(t_1,t_2)} + \frac{k(1-k)Q}{2b_j^{\max}(t_1,t_2)} \tag{4.28}$$

is a valid inequality when $(k-1)Q < f_j(t_1,t_2) - I_{j,t_1-1} \leq kQ$. Finally, note that the above inequality is tightest for k such that $(k-1)Q < f_j(t_1,t_2) - I_{j,t_1-1} \leq kQ$ and thus (4.28) is also valid for all $k = 1, 2, 3, \ldots$.

Analogous reasoning can be used to justify the cut

$$\sum_{v \in V} \sum_{e \in A_v^+(N_v(j,(t_1,t_2)))} (d_e - t_1 + 1) x_e \geq \frac{k\left(f_j(t_1,t_2) - Q_j(t_1,t_2) + I_{j,t_1-1}\right)}{b_j^{\max}(t_1,t_2)} + \frac{k(1-k)Q}{2b_j^{\max}(t_1,t_2)} \tag{4.29}$$

for some discharge port $j \in J_D$, time interval $[t_1, t_2]$, and all $k = 1, 2, 3, \ldots$. Here $f_j(t_1,t_2) - Q_j(t_1,t_2) + I_{j,t_1-1}$ is the amount of inventory discharged in excess of available capacity, and $b_j^{\max}(t_1,t_2)$ is a constant overestimation of the demand rate at j during $[t_1, t_2]$.

Note that inequalities (4.28) and (4.29) are implied by the linear relaxation of the master problem when $k = 1$. Furthermore, since these inequalities are tightest when $(k-1)Q < f_j(t_1,t_2) - I_{j,t_1-1} \leq kQ$ and $(k-1)Q < f_j(t_1,t_2) - Q_j(t_1,t_2) + I_{j,t_1-1} \leq kQ$ respectively, we need only check violation for

$$k = \left\lceil \frac{f_j(t_1,t_2) - I_{j,t_1-1}}{Q} \right\rceil > 2,$$

and

$$k = \left\lceil \frac{f_j(t_1,t_2) - Q_j(t_1,t_2) + I_{j,t_1-1}}{Q} \right\rceil > 2$$

respectively. Separation then is trivial and carried out at each node in the branch-and-bound tree until no appreciable number of such violated cuts are generated.

4.3.3 Branching Decisions

The integrality conditions (4.6) require the columns for each vessel to have the same load/discharge pattern. Thus, a natural branching scheme is to either force a vessel v

to load/discharge at a port j and time point t (the 1-branch), or not load/discharge at j and t (the 0-branch). This is equivalent to branching in the space of the compact arc flow formulation and can easily be enforced by modifying the structure of the network. For the 0-branch, we simply remove from $\vec{\mathcal{N}}_v$ the outgoing arcs from node $n = (j, t)$ corresponding to loading/discharging at n and we do not concatenate labels via arcs with tail at n. The 1-branch can be enforced by first removing nodes $n = (j', t)$ such that $j \neq j'$ (i.e. remove all other ports at the same time point), and remove all arcs $e = ((j_1, t_1), (j_2, t_2))$ such that $t_1 < t$, and $t_2 > t$ (i.e. remove all arcs that bypass time t). This forces every source-sink path to visit the node $n = (j, t)$. Thus, all that remains is to remove from $\vec{\mathcal{N}}_v$ the outgoing arcs from n that correspond to not loading/discharging at n. Although this branching scheme is complete (i.e. sufficient to enforce integrality), it leads to a highly unbalanced tree. Indeed the 1-branch leads to considerable changes whereas the 0-branch leads to very similar fractional solutions.

In a fractional solution, we have a vessel v, port j, and time t such that only some of the fractional columns of v load/discharge at j and t. The remaining fractional columns are either (1) at another port at time t, (2) in transit at time t, or (3) at j during t but do not load/discharge any product. If all remaining fractional columns for v correspond to the third scenario, then the aforementioned branching scheme can be used to eliminate this particular fractional load/discharge decision for v at j and t. Fortunately, this is rarely the case as we almost always have fractional columns that correspond to the first two scenarios leading to fractional solutions that can be eliminated through more effective branching schemes.

For a vessel v, and time point t, we can partition the set of arcs that cross time t by partitioning the set of ports corresponding to the tail node of these arcs, i.e. for a given partition J_0 and J_1 of ports, on one branch we remove all arcs $e = ((j_1, t_1), (j_2, t_2))$ such that $j_1 \in J_0$, $t_1 \leq t$, and $t_2 > t$, and in the other branch remove all arcs

$e = ((j_1, t_1), (j_2, t_2))$ such that $j_1 \in J_1$, $t_1 \leq t$, and $t_2 > t$. Similarly, we can partition the set of arcs that cross time t by partitioning the time point corresponding to the tail node of these arcs, i.e. for a given time point $t' < t$, on one branch we remove all arcs $e = ((j_1, t_1), (j_2, t_2))$ such that $t_1 \leq t$, $t_2 > t$, and $t_1 \leq t'$ and in the other branch remove all arcs $e = ((j_1, t_1), (j_2, t_2))$ such that $t_1 \leq t$, $t_2 > t$, and $t_1 > t'$.

Although an improvement over the first in terms of achieving a more balanced tree, this branching scheme is time dependent and thus often leads to similar fractional solutions after the branch that are simply shifted slightly in time. As a result, several successive branchings have to be made before observing any significant improvement in the bound. To avoid this situation, we also consider branchings that partition the set of ports that are visited first and last. Furthermore, if all loads precede all discharges (which is typical in maritime transportation due to the large distance between load and discharge ports), we can also branch by partitioning the set of ports at which we load last and discharge first. Partitioning the location of the first/last load/discharge can be enforced by removing the appropriate arcs from the source, to the sink, or between appropriate load-discharge pairs. Similarly, we can also branch on the timing of the first/last load/discharge and enforce these by also removing the appropriate arcs from the source, to the sink, or between appropriate load-discharge pairs. Fixing the location and/or timing of the first/last load/discharge typically imposes considerable structure for the remainder of the route leading to immediate improvements in bounds. Furthermore, it alleviates some of the time symmetry when the time dependent branching schemes mentioned earlier are used thereafter.

Finally, if it is required that a vessel visits a port at most once (although it is free to load/discharge multiple times during a visit) then we can use a "follow-on" type branching scheme that is often used in crew scheduling (see Ryan and Foster [96]) to partition the set of ports a vessel visits next. Here for a given port j, we partition the set of ports a vessel can relocate to from j. In one branch we only allow relocations

from j to ports in J_0, and only allow relocations to ports in J_1 in the other branch, where (J_0, J_1) is a partition of the ports not including j. This branching scheme allows us to remove arcs across the time horizon leading to a branching scheme with a typically much larger impact on bound than any of the previously mentioned schemes.

Since we are mainly interested in obtaining good bounds from our branch-price-and-cut approach, we employ a best bound search within the branch-and-bound tree. Furthermore, when choosing a branching decision to eliminate a fractional solution, we look for branching dichotomies in the reverse order of the schemes introduced in this section, since these branches are much more effective than the former in terms of bound improvement.

4.4 Computational Experiments

We test our branch-price-and-cut algorithm on practical instances related to managing the supply chain of an intermediary petrochemical product for a large oil company. Market demands dictate that this product is shipped from supply points in Europe to refineries in the US. The travel times between supply points in Europe or between demand points in the US are typically no more than a few days whereas the transatlantic journey typically takes over two weeks. A daily time discretization over two month planning horizon is considered for each instance. Physical limitations such as draft limits mean that 2 classes of vessels are typically used with capacities around 60, and 80 deadweight kilotons respectively. Each individual vessel has its own time window (typically stated as a two week window during which all loading must be completed) and its own cost structure. The total amount of inventory produced at supply ports in excess of total storage capacity is approximately the same as the total amount of inventory demanded in excess of what is available at demand ports, i.e. $\sum_{j \in J_S} (I_{j,0} + b_j(1,T) - Q_{j,T})^+ \approx \sum_{j \in J_D} (b_j(1,T) - I_{j,0})^+$. Furthermore, this total excess/deficit accounts for about half of the fleet's shipping capacity. Thus, the usage

of about half of the fleet's capacity is dictated by feasibility requirements, and the remaining half by costs/profitability. Finally, port capacities and production/demand rates may fluctuate but typically remain constant for at least a few days at a time.

Within the above parameters, we test our algorithm on 35 instances with varying number of vessels and number of supply/demand points. An instance denoted by $(6, 3, 4, 1)$ for example corresponds to the first instance with 6 vessels, 3 load ports, and 4 discharge ports. We also test our algorithm against the compact arc flow formulation described in Song and Furman [101] (an extension of the work of Savelsbergh and Song [99]) using their branch-and-cut approach within CPLEX 11.1. All experiments were conducted on 2.66GHz Intel Xenon CPUs with 4GB of RAM.

4.4.1 Impact of Preprocessing and Cuts on the Linear Relaxation

We first conduct experiments to evaluate the relative impact of the preprocessing and various cuts introduced in Section 4.3.2. For each of the 35 instances, we solve the root relaxation without any preprocessing or adding any cuts. We then incrementally introduce preprocessing and cuts and observe the impact on the bound and the time it takes to solve the LP relaxation. Tables 4.1 and 4.2 summarize the results of these experiments over all 35 instances. The first row in Tables 4.1 and 4.2 lists the instances grouped by the number of vessels, load ports and discharge ports, and the second row displays the number of instances within each of these groups. The third row in Table 4.1 displays the average gap between the root relaxation and the best known integer solution for instances within the same group. Rows four to eight display the average portion of this gap that is closed after incrementally including preprocessing, the boundary constraints within the DP, port capacity cuts, vessel capacity cuts, and the timing cuts respectively. Similarly, the third row in Table 4.2 displays the average time to solve the root relaxation for instances within the same group. Rows four to eight display the average time after incrementally including

preprocessing, the boundary constraints within the DP, port capacity cuts, vessel capacity cuts, and the timing cuts respectively.

From Table 4.1 it is clear that the preprocessing, boundary constraints, and cuts all contribute significantly to closing the gap at the root node. Indeed, 40% to 70% of the LP gap is closed by including preprocessing, boundary constraints, and various cuts. However, this does come at the expense of taking longer to solve the respective LP relaxations (see Table 4.2). Note that time spent in separating violated inequalities is negligible and thus, the additional time required to solve the relaxation after adding cuts corresponds to the time spent in the additional pricing iterations after each round of adding cuts. Although there is a sharp increase in the time taken to solve the LP relaxation after including the timing cuts, the incremental improvement in the bound after including these cuts is also significant compared to the gap closed by other means.

4.4.2 Integer Solution and Bound Results

In this section, we report the results for our branch-price-and-cut algorithm and compare the bounds produced to bounds obtained from the branch-and-cut algorithm of Song and Furman [101]. For each of the 35 instances, Table 4.3 reports the lower bound LB, the upper bound UB, gap, the time to solve (in seconds), and the number of nodes in the branch-and-bound tree using each of the two methods, namely our branch-price-and-cut algorithm (BP&C), and the branch-and-cut algorithm of Song and Furman (B&C). The last column reports the gap to best upper bound obtained either during our branch-price-and-cut algorithm or the branch-and-cut algorithm of Song and Furman. A time limit of 24 hours was given to solve each instance.

Of the 35 instances, our branch-price-and-cut algorithm solves 20 instances within the 24 hour time limit compared to 16 instances using the branch-and-cut algorithm of Song and Furman. Of the instances that could be solved by both methods, our

approach was on average 6 times faster (5 times faster for instances with 4 vessels, and 7 times faster for instances with 5 vessels). Furthermore, of the 19 instances that could not be solved by the branch-and-cut approach, our branch-price-and-cut algorithm produces an average gap of 6% to the best upper bound, compared to 44% for the branch-and-cut algorithm. Note however that although we produce much better lower bounds, our approach fails to find a feasible solution for several of the larger instances. Indeed, since we use a best-bound strategy to search the branch-and-bound tree, we only obtain solutions (i.e. upper bounds) close to the end of the branch-and-bound process.

4.5 Conclusions

In this chapter we have introduced a new branch-price-and-cut algorithm for IRP that considers a more complex routing component than previous time indexed formulations, and does away with assumptions regarding constant capacities and constant production/demand rates as compared to other column generation formulations. The column generation pricing problem relies on the solution of a mixed integer problem. We extend the DP algorithm developed for FCSPP in Chapter 3 to solve this pricing problem efficiently. In addition to preprocessing, we use the boundary constraints to restrict the set of columns that are generated. Furthermore, we extend a class of cuts known for the vehicle routing problem, and develop a new class of cuts specifically for IRP to tighten the formulation even further.

Our computational results indicate that our branch-price-and-cut algorithm outperforms the branch-and-cut algorithm of Song and Furman [101] with respect to the time taken to solve individual instances as well as the lower bounds produced for difficult instances that could not be solved within 24 hours.

Although we are able to produce meaningful bounds for difficult instances, there is still room for improving both the primal and dual bounds. On the dual side, the

existing cuts can be strengthened by considering cuts that only include vessels with similar capacities. Furthermore, the fact that the timing cuts significantly improve the bounds even after adding the full arsenal of port capacity and vessel capacity cuts is an encouraging indication that we may be able to identify new mixed integer cuts to further strengthen the formulation. On the primal side, given the large number of nodes processed, it would be unwise to rely solely on the branch-and-bound process to produce good upper bounds. Column generation based heuristics and local search techniques have proven useful in providing good feasible solutions to other difficult problems and show promise here as well.

Table 4.1: The impact of preprocessing, boundary constraints, and cuts on the bound of the linear relaxation.

Instance:	$(4,2,2,*)$	$(5,3,2,*)$	$(5,2,3,*)$	$(5,3,3,*)$	$(6,3,4,*)$	$(6,4,3,*)$	$(6,4,4,*)$
No. of instances	5	5	5	5	5	5	5
Average root LP % gap	50.7	142.0	114.3	25.5	58.0	50.1	20.0
Average % gap closed after:							
+ Preprocessing	8.8	7.3	1.6	16.7	16.2	13.4	17.4
+ Boundary Constraints	10.6	8.2	4.6	23.2	18.6	15.8	26.8
+ Port Capacity Cuts	32.8	10.3	7.0	27.7	27.3	18.1	40.1
+ Vessel Capacity Cuts	42.4	14.4	8.4	32.1	30.0	20.5	45.3
+ Timing Cuts	69.3	67.8	44.2	59.4	69.3	49.8	64.1

Table 4.2: The impact of preprocessing, boundary constraints, and cuts on the time to solve the linear relaxation.

Instance:	$(4,2,2,*)$	$(5,3,2,*)$	$(5,2,3,*)$	$(5,3,3,*)$	$(6,3,4,*)$	$(6,4,3,*)$	$(6,4,4,*)$
No. of instances	5	5	5	5	5	5	5
Average root time (s)	0.5	3.0	5.0	8.6	32.0	39.4	43.0
Average time (s) after:							
+ Preprocessing	0.4	4.3	6.4	16.6	108.4	62.5	177.0
+ Boundary constraints	0.5	5.8	7.3	21.4	138.6	77.8	210.6
+ Port capacity cuts	0.8	8.2	13.9	30.2	227.0	102.0	344.4
+ Vessel capacity cuts	0.8	8.0	14.1	33.1	212.7	126.9	331.5
+ Timing cuts	3.4	53.7	49.7	156.6	951.6	510.1	915.4

Table 4.3: Integer solutions and bounds for all instances.

Instance	Method	LB	UB	% gap	Time (s)	Nodes	% gap*
(4,2,2,1)	BP&C	-2867	-2867	0.0	108	151	0.0
	B&C	-2867	-2867	0.0	112	4,751	0.0
(4,2,2,2)	BP&C	-3576	-3576	0.0	7	37	0.0
	B&C	-3576	-3576	0.0	9	383	0.0
(4,2,2,3)	BP&C	596	596	0.0	1	15	0.0
	B&C	596	596	0.0	2	43	0.0
(4,2,2,4)	BP&C	-2469	-2469	0.0	1	15	0.0
	B&C	-2469	-2469	0.0	20	777	0.0
(4,2,2,5)	BP&C	1099	1099	0.0	15	35	0.0
	B&C	1099	1099	0.0	97	2,511	0.0
(5,2,3,1)	BP&C	-468	-468	0.0	376	169	0.0
	B&C	-468	-468	0.0	2,129	16,955	0.0
(5,2,3,2)	BP&C	1457	1457	0.0	40	37	0.0
	B&C	1457	1457	0.0	201	1,150	0.0
(5,2,3,3)	BP&C	-329	-329	0.0	2,323	509	0.0
	B&C	-329	-329	0.0	15,563	67,902	0.0
(5,2,3,4)	BP&C	-1924	-1924	0.0	11,247	2,987	0.0
	B&C	-2693	-1804	49.3	TL	305,609	40.0
(5,2,3,5)	BP&C	-436	-436	0.0	183	81	0.0
	B&C	-436	-436	0.0	2,116	13,490	0.0

TL – 24 hour time limit reached

Continued on Next Page...

Instance	Method	LB	UB	% gap	Time (s)	Nodes	% gap*
(5,3,2,1)	BP&C	1679	1679	0.0	2,648	1,659	0.0
	B&C	1679	1679	0.0	9,255	40,366	0.0
(5,3,2,2)	BP&C	304	304	0.0	19,712	9,333	0.0
	B&C	-265	309	185.6	TL	445,098	187.1
(5,3,2,3)	BP&C	2417	2417	0.0	8,617	2,419	0.0
	B&C	2417	2417	0.0	53,019	306,011	0.0
(5,3,2,4)	BP&C	-1478	-1478	0.0	1,883	1,203	0.0
	B&C	-1478	-1478	0.0	5,793	35,694	0.0
(5,3,2,5)	BP&C	2585	2585	0.0	198	729	0.0
	B&C	2585	2585	0.0	23,598	114,920	0.0
(5,3,3,1)	BP&C	-4787	-4705	1.7	TL	13,232	1.7
	B&C	-5314	-4579	16.1	TL	322,574	12.9
(5,3,3,2)	BP&C	-5101	-5101	0.0	2,495	2,582	0.0
	B&C	-5101	-5101	0.0	49,737	558,868	0.0
(5,3,3,3)	BP&C	-4112	-4112	0.0	4,911	7,788	0.0
	B&C	-4112	-4112	0.0	10,226	129,670	0.0
(5,3,3,4)	BP&C	-5669	-5669	0.0	15,764	7,394	0.0
	B&C	-6087	-5649	7.7	TL	387,087	7.4
(5,3,3,5)	BP&C	-2731	-2416	13.0	TL	10,666	13.0
	B&C	-3643	-2273	60.3	TL	533,467	50.8
(6,3,4,1)	BP&C	-3034	-3034	0.0	15,816	7,458	0.0
	B&C	-3034	-3034	0.0	25,069	165,643	0.0

TL – 24 hour time limit reached

Continued on Next Page...

Instance	Method	LB	UB	% gap	Time (s)	Nodes	% gap*
(6,3,4,2)	BP&C	-1231	-1194	3.0	TL	20,857	3.0
	B&C	-1635	-1194	36.9	TL	504,395	36.9
(6,3,4,3)	BP&C	-4545	-4428	2.6	TL	19,523	2.6
	B&C	-5522	-4196	31.6	TL	372,038	24.7
(6,3,4,4)	BP&C	-3789	-	-	TL	28,092	10.1
	B&C	-4477	-3441	30.1	TL	389,853	30.1
(6,3,4,5)	BP&C	-3076	-	-	TL	25,662	9.8
	B&C	-3799	-2802	35.6	TL	363,321	35.6
(6,4,3,1)	BP&C	-2874	-2656	8.2	TL	22,582	8.2
	B&C	-4493	-2451	83.3	TL	354,250	69.1
(6,4,3,2)	BP&C	-3227	-2887	11.8	TL	22,310	8.2
	B&C	-3916	-2981	31.4	TL	353,688	31.4
(6,4,3,3)	BP&C	-3232	-	-	TL	23,251	15.0
	B&C	-4379	-2811	55.8	TL	378,264	55.8
(6,4,3,4)	BP&C	-5019	-4887	2.7	TL	32,323	2.7
	B&C	-5994	-4857	23.4	TL	569,399	22.6
(6,4,3,5)	BP&C	-2670	-	-	TL	22,489	21.9
	B&C	-4440	-2190	102.7	TL	314,701	102.7
(6,4,4,1)	BP&C	-5123	-	-	TL	14,934	8.1
	B&C	-6461	-4741	36.3	TL	186,286	36.3
(6,4,4,2)	BP&C	-5536	-	-	TL	19,301	7.2
	B&C	-6727	-5166	30.2	TL	240,007	30.2

TL – 24 hour time limit reached

Continued on Next Page...

Instance	Method	LB	UB	% gap	Time (s)	Nodes	% gap*
(6,4,4,3)	BP&C	-4224	-4204	0.5	TL	28,599	0.5
	B&C	-5286	-4166	26.9	TL	266,062	25.8
(6,4,4,4)	BP&C	-4961	-4938	0.5	TL	16,705	0.5
	B&C	-5992	-4623	29.6	TL	212,270	21.3
(6,4,4,5)	BP&C	-4811	-4811	0.0	74,140	48,790	0.0
	B&C	-5392	-4759	13.3	TL	294,065	12.1

TL – 24 hour time limit reached

CHAPTER V

CONCLUSIONS AND FURTHER RESEARCH

In this thesis, we have developed new dynamic programming techniques to solve some difficult pricing problems related to *resource constrained shortest path problems* (RCSPPs). This includes work on the dial-a-flight problem where the sheer size of the networks and number of resource constraints makes the resulting pricing problem intractable using standard dynamic programming techniques for RCSPP, as well as work on the *fixed charge shortest path problem* (FCSPP), a difficult variant of RCSPP with applications in vehicle routing problems with split deliveries, machine scheduling problems with controllable processing times, and inventory routing problems. We have also developed a new branch-price-and-cut algorithm for the inventory routing problem, a difficult mixed 0-1 problem that requires considerable effort in preprocessing, branching, and cut generation outside the realm of branch-price-and-cut for traditional 0-1 problems.

Standard dynamic programming techniques for RCSPP struggle when faced with large time expanded formulations with many resource constraints. However, by using a relaxation-based dynamic programming algorithm that alternates between a forward and a backward search, and by being selective of the resources and arcs we use to refine the relaxation, we are able to maintain a small state space and prune many paths while ensuring an optimal solution to RCSPP is ultimately found. When tested on practical instances of the dial-a-flight problem, we are several orders of magnitude faster than standard dynamic programming techniques. Perhaps more importantly, we can also solve much larger pricing problems than standard dynamic programming techniques that tend to exhaust computer memory more readily. The success resulting

from solving larger pricing problems translates to proving near optimal bounds for instances of the dial-a-flight problem that are otherwise impossible to obtain.

Although we have only tested these ideas for large time expanded and acyclic networks, many of the ideas, in particular the bounding scheme with alternating search directions, can be applied more generally in the context of RCSPP. This requires the development of a classification scheme for resource constraints that can be used to identify resources and arcs over which the relaxation should be tightened to achieve the best compromise between obtaining strong bounds to prune the search and maintaining a small state space. With cyclic networks, we also have to deal with the additional complexity of ensuring the chosen relaxation does not lead to an unbounded problem.

In our study of FCSPP, we obtain a useful characterization of optimal solutions that is exploited in a dynamic programming algorithm. By decomposing the problem into several easier RCSPPs, we are able to solve FCSPP much faster than more naive approaches. On all three classes of problems that we tested, we outperform alternative algorithms, although as predicted, the performance varies depending on the relative size of the bounds allowed on resource consumption.

An extension of the ideas exploring general nested constraints within shortest paths rather than the simple nested structure of the multi-period knapsack problem seems possible. This could lead to column generation formulations of new classes problems where resource consumption is calculated starting from various points along a path rather than always accumulating starting with the source node.

In the final part of the thesis, we present a new branch-price-and-cut algorithm for the *inventory routing problem* (IRP). Apart from considering a more general problem than previously considered in the literature, most of the developments here relate to extending the use of column generation and branch-price-and-cut to the mixed 0-1 case. This includes solving a mixed 0-1 pricing problem, extending 0-1 cuts for

the vehicle routing problem to the mixed 0-1 case, and developing a new class of mixed 0-1 cuts specifically for IRP. Our computational results demonstrate that our branch-price-and-cut algorithm outperforms alternative branch-and-cut algorithms with respect to the time taken to solve individual instances as well as the lower bounds produced for difficult instances that could not be solved within 24 hours.

Although we are able to produce meaningful bounds for difficult instances, there is still room for improving both the primal and dual bounds. On the dual side, it may be possible to strengthen the existing classes of cuts by disaggregating these cuts into groups of cuts for similar vessels. Furthermore, the fact that our new class of cuts significantly improves the bounds, even after adding all other cuts, is an encouraging indication that further investigation may lead to identifying new mixed integer cuts to further strengthen the formulation. On the primal side, column generation based heuristics and local search techniques that have proved useful in providing good feasible solutions to other difficult problems may also prove valuable in providing good upper bounds to complement the tighter lower bounds produced by our branch-price-and-cut approach for harder instances.

REFERENCES

[1] Adelman, D., "Price-directed replenishment of subsets: Methodology and its application to inventory routing," *Manufacturing and Service Operations Management*, vol. 5, pp. 348–371, 2003.

[2] Agarwal, Y., Mathur, K., and Salkin, H. M., "A set-partitioning-based exact algorithm for the vehicle routing problem," *Networks*, vol. 19, pp. 731–749, 1989.

[3] Anbil, R., Tanga, R., and Johnson, E. L., "A global approach to crew-pairing optimization," *IBM Syst. J.*, vol. 31, pp. 71–78, 1992.

[4] Appelgren, L. H., "A column generation algorithm for a ship scheduling problem," *Transportation Science*, vol. 3, pp. 53–68, 1969.

[5] Appelgren, L. H., "Integer programming methods for a vessel scheduling problem," *Transportation Science*, vol. 5, pp. 64–78, 1971.

[6] Balakrishnan, A. and Geunes, J., "Production planning with flexible product specifications: An application to specialty steel manufacturing," *Operations Research*, vol. 51, pp. 94–112, 2003.

[7] Barahona, F. and Anbil, R., "The volume algorithm: producing primal solutions with a subgradient method," *Mathematical Programming*, vol. 87, pp. 385–399, 2000.

[8] Bard, J. F., Huang, L., Jaillet, P., and Dror, M., "A decomposition approach to the inventory routing problem with satellite facilities," *Transportation Science*, vol. 32, pp. 189–203, 1998.

[9] Barnhart, C., Boland, N. L., Clarke, L. W., Johnson, E. L., Nemhauser, G. L., and Shenoi, R. G., "Flight string models for aircraft fleeting and routing," *Transportation Science*, vol. 32, pp. 208–220, 1998.

[10] Barnhart, C., Hane, C. A., and Vance, P. H., "Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems," *Operations Research*, vol. 48, pp. 318–326, 2000.

[11] Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., and Vance, P. H., "Branch-and-price: Column generation for solving huge integer programs," *Operations Research*, vol. 46, pp. 316–329, 1998.

[12] Beasley, J. E. and Christofides, N., "An algorithm for the resource constrained shortest path problem," *Networks*, vol. 19, pp. 379–394, 1989.

[13] Bell, W. J., Dalberto, L. M., Fisher, M. L., Greenfield, A. J., Jaikumar, R., Kedia, P., Mack, R. G., and Prutzman, P. J., "Improving the distribution of industrial gases with an on-line computerized routing and scheduling optimizer," *Interfaces*, vol. 13, pp. 4–23, 1983.

[14] Bellman, R., "On the theory of dynamic programming," *Proceedings of the National Academy of Science*, vol. 38, pp. 716–719, 1952.

[15] Bellman, R., *Dynamic Programming*. Princeton University Press, 1957.

[16] Ben Amor, H. M. T., Desrosiers, J., and Frangioni, A., "On the choice of explicit stabilizing terms in column generation," *Discrete Applied Mathematics*, vol. 157, pp. 1167–1184, 2009.

[17] Bertazzi, L., Savelsbergh, M. W. P., and Speranza, M. G., "Inventory routing," in *The Vehicle Routing Problem: Latest Advances and New Challenges* (Golden, B., Raghavan, S., and Wasil, E., eds.), vol. 43, Springer US', 2008.

[18] Boland, N., Dethridge, J., and Dumitrescu, I., "Accelerated label setting algorithms for the elementary resource constrained shortest path problem," *Operations Research Letters*, vol. 34, pp. 58–68, 2006.

[19] Borndörfer, R., Grötschel, M., and Pfetsch, M. E., "A column-generation approach to line planning in public transport," Tech. Rep. ZIB-Report 05-18, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustraβe 7, Berlin-Dahlem Germany, 2006.

[20] Briant, O., Lemaréchal, C., Meurdesoif, P., Michel, S., Perrot, N., and Vanderbeck, F., "Comparison of bundle and classical column generation," *Mathematical Programming*, vol. 113, pp. 299–344, 2008.

[21] Campbell, A., Clarke, L., Kleywegt, A., and Savelsbergh, M. W. P., "The inventory routing problem," in *In Fleet Management and Logistics* (Crainic, T. G. and Laporte, G., eds.), pp. 95–113, Kluwer Academic Publishers, 1998.

[22] Campbell, A. M., Clarke, L. W., and Savelsbergh, M. W. P., "Inventory routing in practice," in *The Vehicle Routing Problem* (Toth, P. and Vigo, D., eds.), vol. 9, pp. 309–330, SIAM Monographs on Discrete Mathematics and Applications, 2002.

[23] Campbell, A. M. and Savelsbergh, M. W. P., "A decomposition approach for the inventory-routing problem," *Transportation Science*, vol. 38, pp. 488–502, 2004.

[24] Chen, Z. and Powell, W. B., "Solving parallel machine scheduling problems by column generation," *Informs Journal on Computing*, vol. 11, pp. 78–94, 1999.

[25] Cheng, T. C., Kovalyov, M. Y., and Shakhlevich, N. V., "Scheduling with controllable release dates and processing times: Makespan minimization," *European Journal of Operational Research*, vol. 175, pp. 751–768, 2006.

[26] Christiansen, M., "Decomposition of a combined inventory and time constrained ship routing problem," *Transportation Science*, vol. 33, pp. 3–16, 1999.

[27] Christiansen, M., Fagerholt, K., Nygreen, B., and Ronen, D., "Maritime Transportation," in *Handbook in Operations Research and Management Science* (Barnhart, C. and Laporte, G., eds.), vol. 14, Elsevier B.V., 2006.

[28] Christiansen, M. and Nygreen, B., "A method for solving ship routing problems with inventory constraints," *Annals of Operations Research*, vol. 81, pp. 357–378, 1998.

[29] Christiansen, M. and Nygreen, B., "Modelling path flows for a combined ship routingand inventory management problem," *Annals of Operations Research*, vol. 82, pp. 391–413, 1998.

[30] Cordeau, J.-F., "A Branch-and-Cut Algorithm for the Dial-a-Ride Problem," *Operations Research*, vol. 54, pp. 573–586, 2006.

[31] Cordeau, J.-F., Desaulniers, J., Solomon, M. M., and Soumis, F., "VRP with Time Windos," in *The Vehicle Routing Problem* (Toth, P. and Vigo, D., eds.), vol. 9, pp. 157–193, Philadelphia, PA: SIAM Monographs on Discrete Mathematics and Applications, 2002.

[32] Cordeau, J.-F. and Laporte, G., "The dial-a-ride problem: models and algorithms," *Annals of Operations Research*, vol. 153, pp. 29–46, 2007.

[33] Cordeau, J.-F., Laporte, G., Potvin, J.-Y., and Savelsbergh, M. W. P., "Transportation on Demand," in *Handbook in Operations Research and Management Science* (Barnhart, C. and Laporte, G., eds.), vol. 14, Elsevier B.V., 2006.

[34] Cordeau, J.-F., Laporte, G., Savelsbergh, M. W. P., and Vigo, D., "Vehicle Routing," in *Handbook in Operations Research and Management Science* (Barnhart, C. and Laporte, G., eds.), vol. 14, Elsevier B.V., 2006.

[35] Cordeau, J.-F., Soumis, F., and Desrosiers, J., "Simultaneous assignment of locomotives and cars to passenger trains," *Operations Research*, vol. 49, pp. 531–548, 2001.

[36] Daniels, R. L. and Sarin, R. K., "Single machine scheduling with controllable processing times and number of jobs tardy," *Operations Research*, vol. 37, pp. 981–984, 1989.

[37] Dantzig, G. B. and Wolfe, P., "Decomposition principle for linear programs," *Operations Research*, vol. 8, pp. 101–111, 1960.

[38] Day, P. R. and Ryan, D. M., "Flight Attendant Rostering for Short-Haul Airline Operations," *Operations Research*, vol. 45, pp. 649–661, 1997.

[39] Desaulniers, G., "Branch-and-price-and-cut for the split delivery vechicle routing problem with time windows," Tech. Rep. G-2008-32, GERAD, Montreal, Canada, 2008.

[40] Desaulniers, G., Desrosiers, J., and Solomon, M. M., eds., *Column Generation*. Kluwer, 2005.

[41] Desaulniers, G., Desrosiers, J., Solomon, M. M., and Soumis, F., "Daily aircraft routing and scheduling," Tech. Rep. G-94-21, GERAD, Montreal, Canada, 1994.

[42] Desrochers, M., "An algorithm for the shortest path problem with resource constraints," Tech. Rep. Les Cahiers du GERAD G-88-27, University of Montreal, Montreal, Canada, 1988.

[43] Desrochers, M. and Soumis, F., "Crew-opt: crew scheduling by column generation," in *Lecture Notes in Economics and Mathematical Systems* (Daduna, J. R. and Wren, A., eds.), vol. 308, pp. 83–90, Springer-Verlag, 1988.

[44] Desrochers, M. and Soumis, F., "A column generation approach to the urban transit crew scheduling problem," *Transportation Science*, vol. 23, pp. 1–13, 1989.

[45] Desrosiers, J., Dumas, Y., Solomon, M., and Soumis, F., "Time constrained routing and scheduling," in *Handbook in Operations Research and Management Science* (Ball, M. O., Magnanti, T. L., and Nemhauser, G. L., eds.), vol. 8, Elsevier Science Pub Co, 1995.

[46] Desrosiers, J., Soumis, F., and Desrochers, M., "Routing with time windows by column generation," *Networks*, vol. 14, pp. 545–565, 1984.

[47] Dror, M., Laporte, G., and Trudeau, P., "Vehicle routing with split deliveries," *Discrete Applied Mathematics*, vol. 50, pp. 239–254, 1994.

[48] du Merle, O., Villeneuve, D., Desrosiers, J., and Hansen, P., "Stabilized column generation," *Discrete Mathematics*, vol. 194, pp. 229–237, 1999.

[49] Dudzinski, K. and Walukiewicz, S., "Exact methods for the knapsack problem and its generalizations," *European Journal of Operational Research*, vol. 28, pp. 3–21, 1987.

[50] Dumitrescu, I. and Boland, N., "Improved preprocesing, labeling and scaling algorithms for the weight constrained shortest path problem," *Networks*, vol. 42, pp. 135–153, 2003.

[51] Elhallaoui, I., Villeneuve, D., Soumis, F., and Desaulniers, G., "Dynamic aggregation of set-partitioning constraints in column generation," *Operations Research*, vol. 53, pp. 632–645, 2005.

[52] Elhedhli, S. and Goffin, J.-L., "The integration of an interior-point cutting plane method within a branch-and-price algorithm," *Mathematical Programming*, vol. 100, pp. 267–294, 2004.

[53] Energy Information Administration, *Annual Energy Review*. 2007. URL: `www.eia.doe.gov/emeu/aer`.

[54] Espinoza, D., Garcia, R., Goycoolea, M., Nemhauser, G. L., and Savelsbergh, M. W. P., "Per-Seat, On-Demand Air Transportation Part I: Problem Description and an Integer Multi-Commodity Flow Model," *Transportation Science. To appear*, 2006.

[55] Faaland, B. H., "The multiperiod knapsack problem," *Operations Research*, vol. 29, pp. 612–616, 1981.

[56] Federgruen, A. and Zipkin, P., "A combined vehicle routing and inventory allocation problem," *Operations Research*, vol. 32, pp. 109–1037, 1984.

[57] Feillet, D., Dejax, P., Gendreau, M., and Gueguen, C., "An exact algorithm for the elementary problem with resource constraints: Vehicle routing problems," *Networks*, vol. 44, pp. 216–229, 2004.

[58] Feillet, D., Gendreau, M., and Rousseau, L. M., "New refinements for the solution of vehicle routing problems with branch and price," Tech. Rep. C7PQMRPO2005-08-X, Center for Research on Transportation, Montreal, 2005.

[59] Fisher, M., Greenfield, A., Jaikumar, R., and Kedia, P., "Real-time scheduling of a bulk delivery fleet: practical application of lagrangean relaxation," tech. rep., University of Pennsylvania, Department of Decision Sciences, The Wharton School, 1982.

[60] Ford, Jr., L. R. and Fulkerson, D. R., "A suggested computation for maximal multi-commodity network flows," *Management Science*, vol. 5, pp. 97–101, 1958.

[61] Fukasawa, R., Longo, H., Lysgaard, J., Poggi de Arãgao, M., Reis, M., Uchoa, E., and Werneck, R. F., "Robust branch-cut-and-price for the capacitated vehicle routing problem," *Mathematical Programming A*, vol. 106, pp. 491–511, 2006.

[62] Gamache, M., Soumis, F., Marquis, G., and Desrosiers, J., "A column generation approach for large-scale aircrew rostering problems," *Operations Research*, vol. 47, pp. 247–263, 1999.

[63] Garcia, R., *Resource Constrained Shortest Paths and Extensions*. PhD thesis, Georgia Institute of Technology, 2009.

[64] Garey, M. R. and Johnson, D. S., *Computer and intractability: A guide to the theory of NP-completeness*. San Francisco: Freeman, 1979.

[65] Gendreau, M., Dejax, P., Feillet, D., and Gueguen, C., "Vehicle routing with time windows and split deliveries," Tech. Rep. 2006-851, Laboratoire Informatique d'Avignon, France, 2006.

[66] Geoffrion, A. M., "Lagrangean relaxation for integer programming," *Mathematical Programming Study*, vol. 2, pp. 82–114, 1974.

[67] Gilmore, P. C. and Gomory, R. E., "A linear programming approach to the cutting-stock problem," *Operational Research*, vol. 9, pp. 849–859, 1961.

[68] Gilmore, P. C. and Gomory, R. E., "A linear programming approach to the cutting-stock problem: Part 2," *Operational Research*, vol. 11, pp. 863–888, 1963.

[69] Glover, F., "Improved linear integer programming formulations of nonlinear integer problems," *Management Science*, vol. 22, pp. 455–460, 1975.

[70] Goffin, J.-L. and Vial, J.-P., "Convex nondifferentiable optimization: a survey focussed on the analytic center cutting plane method," Tech. Rep. 99.2, Ecole des Hautes Etudes Commerciales, Universite de Geneve, 1999.

[71] Gomory, R. E., "An algorithm for the mixed integer problem," Tech. Rep. RM-2597, The Rand Corporation, 1960.

[72] Haase, K., Desaulniers, G., and Desrosiers, J., "Simultaneous vehicle and crew scheduling in urban mass transit systems," *Transportation Science*, vol. 35, pp. 286–303, 2001.

[73] Handler, G. Y. and Zang, I., "A dual algorithm for the constrained shortest path problem," *Networks*, vol. 10, pp. 293–309, 1980.

[74] Hassin, R., "Approximation schemes for the restricted shortest path problem," *Mathematics of Operations Research*, vol. 17, pp. 36–42, 1992.

[75] Holmberg, K. and Yuan, D., "A multicommodity network-flow problem with side constraints on paths solved by column generation," *Informs Journal on Computing*, vol. 15, pp. 42–57, 2003.

[76] Huisman, D., Jans, R., M., P., and Wagelmans, A., "Combining column generation and lagrangian relaxation," in *Column Generation* (Desaulniers, G., Desrosiers, J., and Solomon, M. M., eds.), ch. 9, Springer, 2005.

[77] Irnich, S. and Desaulniers, G., "Shortest path problems with resource constraints," in *Column Generation* (Desaulniers, G., Desrosiers, J., and Solomon, M. M., eds.), ch. 2, Kluwer, 2005.

[78] Irnich, S. and Villeneuve, D., "The shortest path problem with resource constraints and k-cycle elimination for $k \geq 3$," *Informs Journal on Computing*, vol. 18, pp. 391–406, 2006.

[79] Kallehauge, B., Larsen, J., and Madsen, O. B. G., "Lagrangian duality applied to the vehicle routing problem with time windows," *Computers and Operations Research*, vol. 33, pp. 1464–1487, 2006.

[80] Lemaréchal, C., Nemirovskii, A., and Nesterov, Y., "New variants of bundle methods," *Mathematical Programming*, vol. 69, pp. 111–147, 1995.

[81] Letchford, A. N. and Salazar-González, J.-J., "Projection results for vehicle routing," *Mathematical Programming*, vol. 105, pp. 251–274, 2006.

[82] Lorenz, D. H. and Raz, D., "A simple efficient approximation scheme for the restricted shortest path problem," *Operations Research Letters*, vol. 28, pp. 213–219, 2001.

[83] Lübbecke, M. E., "Dual variable based fathoming in dynamic programs for column generation," *European Journal of Operational Research*, vol. 162, pp. 122–125, 2003.

[84] Lysgaard, J., Letchford, A. N., and Eglese, R. W., "A new branch-and-cut algorithm for the capacitated vehicle routing problem," *Mathematical Programming*, vol. 100, p. 2004, 2003.

[85] Mehlhorn, K. and Ziegelmann, M., "Resource constraint shortest paths," in *ESA2000: Proceedings of the 7th Annual European Symposium on Algorithms*, pp. 326–337, 2000.

[86] Mehrotra, A. and A., T. M., "A column generation approach for graph coloring," *Informs Journal on Computing*, vol. 8, pp. 344–354, 1996.

[87] Nowicki, E. and Zdrzalka, S., "A survey of results for sequencing problems with controllable processing times," *Discrete Optimization*, vol. 26, pp. 271–287, 1990.

[88] Parker, M. and Ryan, J., "A column generation algorithm for bandwidth packing," *Telecommunication Systems*, vol. 2, pp. 185–195, 1994.

[89] Poggi de Aragão, M. and Uchoa, E., "Integer program reformulation for robust branch-and-cut-and-price algorithms," in *Annals of Mathematical Programming in Rio*, (Búzios, Brazil), pp. 56–61, 2003.

[90] Ribeiro, C. C. and Soumis, F., "A column generation approach to the multiple-depot vehicle scheduling problem," *Operations Research*, vol. 42, pp. 41–52, 1994.

[91] Righini, G. and Salani, M., "New dynamic programming algorithms for the resource-constrained elementary shortest path problem," tech. rep., Dipartimento di Tecnologie dell'Informazione, Universita degli Studi di Milano, Via Bramante 65, 26013 Crema, Italy, 2005.

[92] Righini, G. and Salani, M., "Symmetry helps: bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints," *Discrete Optimization*, vol. 3, pp. 255–273, 2006.

[93] Rodrigue, J.-P. and Comtois, C. Slack, B., *The geography of transport systems*. New York: Routledge, 2006.

[94] Ropke, S. and Cordeau, J.-F., "Branch-and-Cut-and-Price for the Pickup and Delivery Problem with Time Windows," Tech. Rep. C7PQMR PO2006-21-X, CRT, Montreal, Canada, 2006.

[95] Ryan, D. M. and Falkner, J. C., "On the integer properties of scheduling set partitioning models," *European Journal of Operational Research*, vol. 35, pp. 442–456, 1988.

[96] Ryan, D. and Foster, B., "An integer programming approach to scheduling," in *Computer Scheduling of Public Transport* (Wren, A., ed.), pp. 269–280, Elsevier, 1981.

[97] Sandhu, R. and Klabjan, D., "Integrated airline fleeting and crew-pairing decisions," *Operations Research*, vol. 55, pp. 439–456, 2007.

[98] Savelsbergh, M. W. P., "A branch-and-price algorithm for the generalized assignment problem," *Operations Research*, vol. 45, pp. 831–841, 1997.

[99] Savelsbergh, M. W. P. and Song, J.-H., "An optimization algorithm for the inventory routing problem with continuous moves," *Computers and Operations Research*, vol. 35, pp. 2266–2282, 2008.

[100] Solomon, M. M., "Algorithms for the vehicle routing and scheduling problem with time window constraints," *Operations Research*, vol. 76, pp. 261–286, 1987.

[101] Song, J.-H. and Furman, K. C., "A maritime inventory routing problem: Practical approach," tech. rep., ExxonMobil Corporate Strategic Research, 2008.

[102] Song, J.-H. and Savelsbergh, M. W. P., "Performance measurement for inventory routing," *Transportation Science*, vol. 41, pp. 44–54, 2007.

[103] Spoorendonk, S., *Cut and column generation*. PhD thesis, DIKU, University of Copenhagen, Denmark, 2008.

[104] Spoorendonk, S. and Desaulniers, G., "Clique inequalities applied to the vehicle routing problem with time windows," Tech. Rep. G-2008-72, GERAD, Montreal, Canada, 2008.

[105] Spoorendonk, S. and Petersen, B., "A branch-and-cut algorithm for the elementary shortest path problem with resource constraints," tech. rep., DIKU Department of Computer Science, Universitesparken 1, DK-2100 Copenhagen $\emptyset$, Denmark, 2005.

[106] Valerio de Carvalho, J. M., "Exact solution of cutting stock problems using column generation and branch-and-bound," *International Transactions in Operational Research*, vol. 5, pp. 35–44, 1998.

[107] Valerio de Carvalho, J. M., "Exact solution of bin-packing problems using column generation and branch-and-bound," *Annals of Operations Research*, vol. 86, pp. 629–659, 1999.

[108] Van den Akker, J. M., Hoogeveen, J. A., and Van de Velde, S. L., "Parallel machine scheduling by column generation," *Operations Research*, vol. 47, pp. 862–872, 1999.

[109] Vance, P. H., *Crew scheduling, cutting stock, and column generation: Solving huge integer programs*. PhD thesis, Georgia Institute of Technology, 1993.

[110] Vance, P. H., Barnhart, C., Johnson, E. L., and Nemhauser, G. L., "Solving binary cutting stock problems by column generation and branch-andbound," *Computational Optimization and Applications*, vol. 3, pp. 111–130, 1994.

[111] Vanderbeck, F., "On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm," *Operations Research*, vol. 48, pp. 111–128, 2000.

[112] Vanderbeck, F., "Implementing mixed integer column generation," in *Column Generation* (Desaulniers, G., Desrosiers, J., and Solomon, M. M., eds.), ch. 12, Kluwer, 2005.

[113] Vanderbeck, F. and Savelsbergh, M. W. P., "A generic view of Dantzig-Wolfe decomposition in mixed integer programming," *Operations Research Letters*, vol. 34, pp. 296–306, 2006.

[114] Xu, H., Chen, Z.-L., Rajgopal, S., and Arunapuram, S., "Solving a practical pickup and delivery problem," *Transportation Science*, vol. 37, pp. 347–364, 2003.